5 Das Turbo Pascal System Version 7.0

Integrierte Entwicklungsumgebung
Integrated Development Environment (IDE)

Gliederung

5.0 Vorbemerkungen

Das Entwicklungsumgebung von Turbo-Pascal (Integrated Development Environment, IDE) enthält im Kern die miteinander verbundenen Funktionen **Editor** (Programm zur Quelltexterstellung), **Compiler** (Übersetzungsprogramm) und einen integrierten **Debugger** (Programm zur Fehlersuche und -behebung). Dazu gibt es Funktionen zur Systemeinstellung der IDE (Optionen) und die gut ausgebaute integrierte **Hilfe-Funktion**.

Die IDE ist in ihrem Leistungsumfang auf den professionellen Pascal-Programmier abgestimmt. Die folgenden Ausführungen zeigen einen Überblick ohne allzusehr ins Detail zu gehen. Der Fortgeschrittene wird sich sicher in der IDE gut zurechtfinden; bei der Arbeit am Rechner ersetzt ihm die gut ausgebaute Hilfe-Funktion die Handbücher.

Somit ist dieses Kapitel in erster Linie für die Anfänger gedacht. Wobei aber viele Punkte erst nach gewissen Lernfortschritten nachgeschlagen werden müssen.

Maus oder Tastenkürzel?

Der Autor dieses Skriptums geht davon aus, daß bei dem gegebenen Zeitpunkt jeder PC mit eines Maus ausgestattet ist und beschreibt vorrangig die Bedienung der IDE mit der Maus. Er gibt aber auch gerne zu, daß in vielen Fällen eine Bedienung mit Tastenkürzeln (Funktionstasten F1 bis F10 oder Tastenkombinationen, hot keys) praktischer ist, zumindest für die Fortgeschrittenen, und verweist auf das von ihm herausgegebene **Hilfsmittel Arbeitsblatt und Tastaturschablone**. Also nicht "Maus *oder* Tastenkürzel", sondern "Maus *und* Tastenkürzel"! Tastenkürzel werden in diesem Skriptum nur nach subjektiver Auswahl des Autors aufgeführt.

5.1 Das Menüsystem der IDE

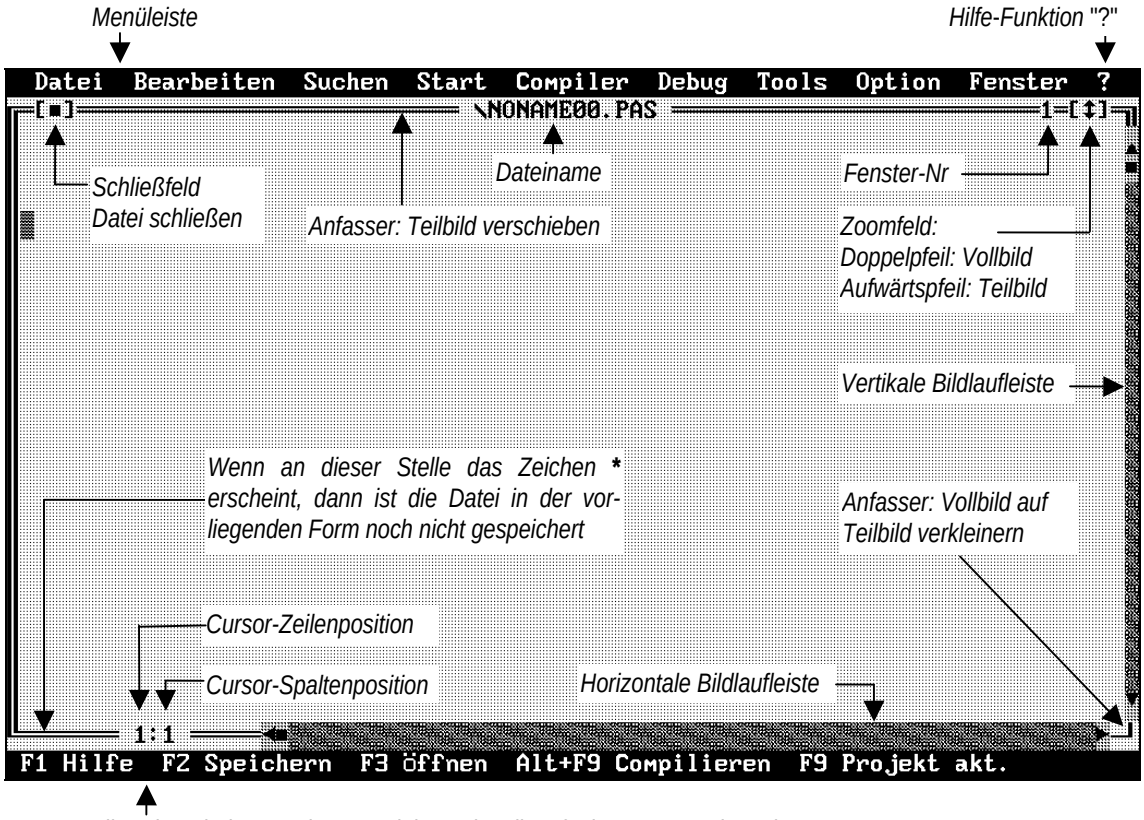

Abbildung 5-B01: Das Menüsystem mit Menüleiste, Editfenster und Statuszeile

Die Menüleiste

Die oberste Bildschirmzeile ist die **Menüleiste** mit den zehn Hauptmenüpunkten, die hier zur besseren Kurzerklärung untereinander geschrieben sind.

1. **Datei** Dateifunktionen
2. **Bearbeiten** Editorfunktionen. Der Editor wird genauer in Kap. 06 behandelt
3. **Suchen** Suchen/Ersetzen, Laufzeitfehler suchen
4. **Start** Einfacher: Programm mit **Strg+F9** compilieren *und* ausführen
5. **Compiler** Nur compilieren
6. **Debug** Integrierter Debugger
7. **Tool** Aufruf von Hilfsprogrammen
8. **Option** Optionen für IDE einstellen
9. **Fenster** Fensterverwaltung
10. **?** Hilfe

Die Menüpunkte können mit Klick auf linke Maustaste angewählt werden oder durch das Tastenkürzel Alt+unterstrichenen Buchstaben in der vorstehenden Liste, z.B. Alt+D (Taste "Alt" gedrückt halten und dann Taste "D" drücken) für "Datei". In der IDE sind die betreffenden Buchstaben (es sind nicht immer die Anfangsbuchstaben des Befehle!) für die Alt-Tastenkürzel nicht durch einen Unterstrich, sondern in roter Farbe (mitunter gelber Farbe) dargestellt; im DOS-Fenster von Windows aber nicht immer gut sichtbar. Diese Festsetzung gilt nicht nur für das (Haupt-) Menü, sondern auch für alle Untermenüs.

Bei einem versichtlich oder falsch angewählten Menüpunkt gelangt man durch einen Mausklick auf eine freie Stelle im Editfenster wieder zu diesem zurück.

Nach Anwahl eines beliebigen Menüpunktes kann zusätzlich mit Klick auf die rechte Maustaste oder mit dem Tastenkürzel **Alt+F10** ein **lokales Menü** aufgerufen werden, das alle im aktuellen Fenster möglichen Befehle anbietet. Mit der Funktionstaste **F1** kann außerdem ein **Hilfetext** zu diesem Menüpunkt aufgerufen werden.

Das Editfenster

Nach der Menüleiste folgt das bis zu 21 Zeilen umfassende **Editfenster**. Links befindet befindet sich das **Schließfeld**, mit dem die Datei des aktuellen Fensters geschlossen werden kann. Das alternative Tastenkürzel für das Schließfeld: **Alt+F3**. Ist die Datei noch nicht gespeichert, erscheint eine Abfrage ob die Datei vor dem Schließen noch abgespeichert (gesichert) werden soll.

In der Mitte des Editfensters erscheint der Dateiname nach MS-DOS-Konvention; bei neu angelegten Dateien der Dummy-Dateiname "NONAME.PAS". Es empfiehlt sich, nach dem Anlegen einer neuen Datei mit den Menüpunkten "Datei/Datei speichern unter..." der Datei einen eigenen Namen zu geben.

In der rechten oberen Ecke des Editfensters wird die Nummer des aktuellen Fensters angezeigt. Es können mehrere Fenster geöffnet sein, das **aktuelle Fenster** (immer nur eines) ist am oberen Rand mit einer **Doppellinie** begrenzt ist. Die Fenster können nebeneinander oder hintereinander (teilweise oder ganz verdeckt) plaziert sein. Rechts neben der Fensternummer befindet sich das **Zoomfeld**, in dem entweder ein **Doppelpfeil** (Fenster ist auf maximale Größe eingestellt, Vollbild, siehe Abbildung) oder ein **Aufwärtspfeil** (Fenster ist verkleinert, Teilbild) angezeigt wird. Mit Mausklick auf die Pfeile kann umgeschaltet werden.

Mit dem **rechts unten** plazierten **Anfasser** kann **die Fenstergröße verändert** werden, von maximal Vollbild mit 21 Zeilen und 78 Spalten auf minimal 3 Zeilen und 21 Spalten. Im Zoomfeld erscheint der Aufwärtspfeil. Ein verkleinertes Fenster kann durch **Anfassen** der **oberen Doppellinie** mit der Maus bei gedrückter Maustaste beliebig auf dem Bildschirm **verschoben** werden. Damit können z.B. auch verdeckte Fenster wieder sichtbar gemacht werden.

Mit den rechts und unten plazierten **Bildlaufleisten** kann der **Bildausschnitt** des aktuellen Fensters beliebig vertikal oder horizontal **gescrollt** werden. Der Profi benutzt für diese Zwecke aber meistens die Richtungstasten (Pfeiltasten, Cursortasten). In der unteren Begrenzung des Editfensters werden die aktuellen Zeilen- und Cursorpositionen angezeigt. Links davon erscheint das Sternzeichen "*", wenn die Datei in der vorliegenden Form noch nicht gespeichert ist. Das Zeichen erscheint auch, wenn bei einer gespeicherten Datei eine Änderung rückgängig gemacht wird, also die Datei im Editor mit der gespeicherten Version identisch ist.

Alle Editfenster erhalten beim Öffnen bzw. Anlegen eine fortlaufende Nummer. Im Rahmen der Speicherkapazität können beliebig viele Editfenster geöffnet werden. Standardmäßig werden die Fenster überlappend angeordnet. Über Menüpunkt "Fenster" kann auch "Nebeneinander" eingestellt werden. In beiden Fällen natürlich nur als Teilbild. Durch Anklicken (eines sichtbaren Bereiches) eines anderen Editfensters wird dieses zum aktuellen Editfenster.

Mit **Alt+0** kann ein Menü aufgerufen werden, das eine Liste aller geöffneter Fenster (nicht nur Editfenster, sondern auch Meldungs-, Debug- und Hilfefenster) angezeigt werden. In dem Menü ist es auch möglich, Fenster zu löschen. Bei Editfenstern kommt bei nicht gespeicherter Datei eine Sicherheitsabfrage. Es ist wegen der visuellen Kontrolle aber sicherer, Editfenster mit dem Schließfeld des Editfensters zu schließen. Mit **Alt+Fensternummer** kann jederzeit auch mit der Tastatur von einem Editfenster auf ein **anderes Editfenster umgeschaltet** werden, z.B. Alt+3 schaltet auf Editfenster Nr. 3 um, unabhängig davon ob dieses vorher sichtbar war oder nicht. Siehe auch Hauptmenüpunkt Fenster, Kap. 5.10.

Die Mausbedienung der IDE ist intuitiv erlernbar. Man klickt die Hauptmenüpunkte einfach an. Es erscheinen dann in vertikaler Anordnung Untermenüs (pull down menu) die ebenfalls angeklickt werden können.

Jeder Untermenüpunkt enthält ein rot geschriebenes Zeichen (nicht gut wahrnehmbar), das in Verbindung mit der Alt-Taste ein Tastenkürzel für Tastaturbedienung ergibt. In *Dialogfenstern* (siehe später) werden für die gleichen Zwecke die betreffenden Zeichen in gut sichtbarer gelber Farbe angeboten. Mitunter sind einige Untermenüpunkte mit reduzierter Helligkeit dargestellt, wobei die Unterscheidung leider auch nicht gut wahrnehmbar ist. Die reduzierte Helligkeit bedeutet, daß die betreffenden Untermenüpunkte in der gegebenen Situation nicht angewählt werden können. Oft zeigen die Untermenüs auch noch weitere Tastenkürzel in Klarform an, z.B. das **Alt+X** im Menüpunkt "Datei" für Beenden der Pascal-Sitzung.

Desweiteren werden häufig **Untermenüpunkte mit nachgesetzten drei Punkten** angeboten, z.B. bei "Datei/Speichern unter...". Die drei Punkte tun kund, daß bei Anwahl ein **Dialogfenster** erscheint, das folgende Elemente enthalten kann:

- **CheckBox**, in Borland Pascal Handbüchern "Markierungsfeld" genannt. Die Bezeichnung "CheckBox" ist in Windowssprachen und auch in Delphi üblich, mitunter

auch die Eindeutschung "Kontrollkästchen". Es spricht nichts dagegen, die Bezeichnung "CheckBox" zu übernehmen. In der IDE wird die CheckBox mit "**[ ]**" gekennzeichnet. Beim Anklicken erscheint ein "X" oder das vorhandene "X" verschwindet. Es können *beliebig viele* der angebotenen CheckBoxen mit "X" aktiviert werden. Beispiel für eine aktivierte CheckBox: "**[**X**]**". Darstellung von CheckBoxen in Windows: Ein kleines Quadrat, bei Aktivierung mit einem X oder ein Häkchen.

- **OptionButton,** in Borland Pascal Handbüchern "Schaltfeld" genannt. In Windowssprachen werden die Bezeichnungen "OptionButton" (VB und VBA) oder "RadioButton (Delphi) benutzt. Der erstgenannte Begriff wird hier bevorzugt. Es werden *immer mehrere OptionButton* in einer "OptionGroup" angeboten. Es kann aber *immer nur ein OptionButton aktiviert* werden. Beim Anklicken eines anderen OptionButton in der gleichen OptionGroup wird der erstangeklickte automatisch deaktiviert. In der IDE werden OptionButtons mit "**( )**" gekennzeichnet.. Beim Anklicken erscheint ein mittiger Punkt. Beispiel für einen aktivierten OptionButton: "**(•)**". Darstellung in Windows: Ein Kreis, bei gewählter Option mit einem Punkt.
- **Eingabefelder** (in Windowssprachen z.B. "EditBox" genannt) für erforderliche Eingaben. Dazu stehen die üblichen Editfunktionen (siehe Kap. 06) zur Verfügung.
- **Listenfelder** (in Windowssprachen z.B. "ListBox" genannt) in denen mit Cursortasten auf/ab oder durch Mausklick auf die angebotenen Pfeilsymbole Elemente ausgewählt werden können. Listenfelder und Eingabefelder können kombiniert auftreten (editierbare Auswahllisten, in Windowssprachen "ComboBox" genannt).
- **Befehlsschaltflächen** (Button) für die Maus wie z.B. "OK", "Abbruch", "Hilfe" usw. Ein Dialogfenster enthält mindestens eine der beiden Schaltflächen "OK" oder "Abbruch". Ausnahme: Einige Unter-Dialogfenster.
- **Bildlaufleisten** vertikal und horizintal. Nur bei Bedarf bei Listenfeldern.
- **Schließfeld** in der linken oberen Ecke des Dialogfeldes. In jedem Fenster vorhanden. Das Schließfeld wirkt wie die Schaltfläche "Abbruch" oder die Taste "Esc".

Die folgende Abbildung zeigt ein typisches Dialogfenster. Es erscheint beim Aufruf des Menüpunktes "Suchen/Ersetzen...". Es hat die Fensterbezeichnung "Text ersetzen" und enthält folgende Elemente: zwei kombinierte Eingabe- und Listenfelder (ComboBoxen) mit den Bezeichnungen "Suchen nach" und "Ersetzen durch", ein Checkbox-Gruppenfeld mit der Bezeichnung "Optionen" mit vier Checkboxen, drei OptionGroups mit den Bezeichnungen "Bereich", "Richtung" und "Beginn" mit je vier OptionButtons, vier Schaltflächen mit den Bezeichnungen "OK", "Alles ersetzen", "Abbruch" und "Hilfe" sowie das obligatorische Schließfeld in der linken oberen Ecke. Dialogfelder können nach den Mausanfassen an der oberen Doppellinie beliebig verschoben werden. Wenn der Ersetzvorgang wiederholt wird, dann können die bisherigen Such- und Ersatzwörter durch Anklicken der Abwärtspfeilsymbole in den Listenfeldern angezeigt werden. Es erscheint ein Unter-Dialogfenster (hier nicht dargestellt) mit den bisherigen Wörtern. Mit Doppelklick auf das gewünschte Wort oder mit der Enter-Taste wird das Wort in das Eingabefeld kopiert und das Unterfenster geschlossen. Das in Frage stehende Unter-Dialogfenster besitzt keine Schaltflächen, sondern nur noch nur ein Schließfeld, von Bildlaufleisten abgesehen.

Abbildung 5-B02: Ein typisches Dialogfenster

Zur Tastaturbedienung der IDE

Mit der Funktiontaste **F10** die **Menüleiste aktivieren**. Dann mit Cursortasten links oder rechts den Leuchtbalken auf den gewünschten Menüpunkt stellen, dann mit Cursortaste unten oder mit Eingabetaste (Enter, Return) auswählen. Es erscheint dann das betreffende Untermenü. Die Auswahl der Untermenüpunkte durch Leuchtbalkenverstellung mit Cursor *unten* oder *oben* und Aufruf mit der Enter-Taste.

In jeder Situation kann mit der Escape-Taste (Taste "Esc" auf der Tastatur links oben) abgebrochen werden. Die Dialogfenster enthalten Maus-Befehlsschaltflächen (Button), z.B. "OK", "Abbruch" und "Hilfe". Der Tastaturersatz für "OK" ist die Enter-Taste, für "Abbruch" die Esc-Taste und für "Hilfe" die Funktionstaste "F1". Mausbedienung und Tastaturbedienung können beliebig kombiniert werden.

Die Statuszeile der IDE

In der Statuszeile werden einige aktuelle Tastenkürzel (für den Anfänger sind nur die Funktionstasten F1, F2 und F3 von Belang) angezeigt. Bei Auswahl eines Menü- oder Untermenüpunktes erscheint in der Statuszeile eine Kurzinformation.

F1 Hilfe. Schnellerer Ersatz für Menüpunkt "**?**" (= Hilfe). Details siehe dort. Mit dieser Funktionstaste **F1** (Hilfe) wird ein kontextbezogener Hilfetext in einem Hilfefenster angezeigt. Wenn gelb hervorgehobene Stichwörter erscheinen, dann kann mit dem Hypertext-Verfahren durch Doppelklick dorthin verzweigt

werden. Die Hilfefenster sind ähnlich wie die Editfenster in der Größe veränderbar und bieten ebenfalls ein Schließfeld an, das alternativ zur Taste Esc benutzt werden kann. Mit **Alt+F1** wird das zuletzt aufgerufene Hilfefenster wieder angezeigt. **Besondes wichtig**: Steht der Cursor auf einem **Pascal-Begriff** (reserviertes Wort, Standardbezeichner für Datentypen, Prozeduren und Funktionen), dann kann mit **Strg+F1** ein **kontextbezogener Hilfetext** zu diesem Begriff aufgerufen werden. Alternative: **Strg+rechteMaustaste**.

F2 Datei speichern. Schnellere Alternative zum Menüpunkt "Datei/Speichern". Details siehe dort".

F3 Datei öffnen (laden). Schnellere Alternative zum Menüpunkt "Datei/Öffnen...". Details siehe dort.

Alt+F9 Compilieren. Schnellere Alternative zum Menüpunkt "Compiler/Compilieren". Details siehe dort. Für den Anfänger ohne Belang.

5.2 Der Hauptmenüpunkt Datei

Abbildung 5-B03: Der Hauptmenüpunkt Datei

Der Hauptmenüpunkt *Datei* enthält folgende Untermenüpunkte (mit den unterstrichenen Zeichen können weitere Alt-Tastenkürzel gebildet werden):

Neu Eine neue Pascal-Quelltextdatei wird angelegt. Sie erhält den Dummy-Dateinamen "NONAME.PAS" und wird beim "Speichern" im aktuellen Verzeichnis abgelegt.

Öffnen... Tastenkürzel "F3". Es erscheint ein Dialogfenster mit einem kombinierten Eingabe- und Listenfeld "Name" und einem weiteren Listenfeld "Datei", in dem alle existierenden Pascal-Dateien des aktuellen Verzeichnisses und die darunter liegenden Unterverzeichnisse zur Auswahl mit Doppelklick angeboten werden. Unterverzeichnisse werden durch einen nachgesetzten Backslash gekennzeichnet. Durch Doppelklick kann man in das gewünschte Unterverzeichnis wechseln. Über das Symbol "..\" kann man auch zu übergeordneten Verzeichnissen wechseln.

Zum Wechseln des Verzeichnisses kann man auch den Untermenüpunkt "Verzeichnis wechseln..." benutzen, mit dem man zusätzlich auch noch das Laufwerk wechseln kann. Wenn man in das Eingabefeld "Name" eine nichtexistierende Datei eintippt, wird eine neue Datei unter diesem Namen angelegt. Auch eine Möglichkeit, eine neue Datei von Anfang an mit einem eigenen Namen statt mit "NONAME.PAS" anzulegen. Im Feld "Name" können auch die Jokerzeichen "*" und "?" benutzt werden. Es werden dann im Listenfeld "Datei" nur die Dateien angezeigt, auf die die Maskierung mit den Jokerzeichen paßt. Die Standardeinstellung ist sinnvollerweise "*.PAS".

Speichern Tastenkürzel "F2". Datei des aktuellen Editfensters wird im aktuellen Verzeichnis unter dem im Editfenster oben angegebenen Dateinamen gespeichert, u.U. noch unter dem Dummy "NONAME.PAS", was nicht sehr klug ist. Deshalb kurz nach dem Anlegen einer neuen Datei mit dem nächsten Menüpunkt "Speichern unter..." einen eigenen Namen vergeben und das gewünschte Verzeichnis auswählen.

Speichern unter... Es erscheint ein Dialogfenster. Wenn in einer Liste untergeordnete Verzeichnisse aufgelistet werden (erkenntlich an einem nachgesetztem Backslash), dann kann man mit Doppelklick zuerst dorthin wechseln. Zum übergeordneten Verzeichnis wechselt man durch Anklicken des Symbols "..\". In einem Eingabefeld kann der gewünschte Dateiname eingetragen werden. Die Extension ".PAS" ist Standard und braucht nicht eingegeben werden; die Eingabe schadet aber auch nicht.

Alles speichern Speichert die Dateien *aller* Editfenster.

Verzeichnis wechseln... Es erscheint ein Dialogfenster mit dem hierarchischen Verzeichnisbaum. Zum gewünschten Verzeichnis gelangt man durch Doppelklick. Evtl. muß man zuerst mit einer Bildlaufleiste vertikal scrollen. Am oberen Ende des Baumes steht "Drives". Nach Doppelklick darauf werden alle verfügbaren Laufwerke angezeigt. Mit Doppelklick wird das gewünschte Laufwerk zum aktuellen Laufwerk. Dann hat man weiter das gewünschte Verzeichnis in der bereits beschriebenen Weise einzustellen.

Drucken Gibt die komplette Datei im aktuellen Editfenster auf den Drucker aus. Leider ohne Zeilen- und Seitennummern. Dafür werden bei geeigneten Druckern und richtiger Installation mit "Druckerinstallation..." die reservierten Wörter fett gedruckt. In der Regel will man nur markierte Teile ausdrucken. Für diesen Fall nach dem Markieren lediglich die Hotkey **Strg+P** benutzen.

Den Studierenden der Druckereitechnik steht auf den Rechnern des Programmierlabors in der IDE unter "Tools" das Programm **Drucke** des Autors zur Verfügung, das auch mit der Hotkey "Umsch+F9" aufgerufen werden kann. Das Programm gestattet u.a. den seitenweisen Ausdruck beliebiger (Text-) Dateien, optional mit Zeilen- und Seitennummern, wählbarer Schriftgröße u.a.m. Für die erforderlichen Eingaben erscheint ein Dialogfenster, in dem u.a. auch der Dateiname eingegeben werden muß. Es können somit nur abgespeicherte Dateien gedruckt werden, deshalb immer vor dem Aufruf von "Drucke" die Datei mit "F2" speichern. Für die

Progammlistungs der Praktikumsaufgaben ist immer "Drucke" zu benutzen; Progammlistings in einer sonst noch so schönen Proportionalschrift sind eine Zumutung.

Druckerinstallation... Vorab: Der Anfänger möge diesen Punkt vorerst übergehen, wenn er mit dem Menüpunkt "Drucken" nichts am Hut oder damit keine Schwierigkeiten hat. Ansonsten: Es erscheint ein Dialogfenster mit den Eingabefeldern "Pfad für Filter" und "Kommandozeilen-Parameter", einer CheckBox "Syntaxhervorhebung", den Schaltflächen "OK" , "Abbruch" und "Hilfe". In das Eingabefeld "Pfad für Filter" ist einzutragen, wenn bei der Installation von Borland Pascal nicht bereits erfolgt: "C:\BP\EXAMPLES\UTILS\PRNFLTR.EXE". Wenn die Datei "PRNFLTR.EXE" in einem anderen Verzeichnis steht, ist der Eintrag entsprechend zu ändern. Zum Druckerfilter "PRNFLTR.EXE" steht im genannten Verzeichnis auch der Pascal-Quelltext "PRNFLTR.PAS" (ca. 13 KByte) zur Verfügung. Diese Datei ist mit Turbo-Pascal editierbar. Sie muß dann zu einem EXE-File compiliert werden, siehe Menüpunkt "Compiler". Wenn man diese Datei ändert, kann man auch einigen Schnickschnack in der Druckerausgabe des Menüpunktes "Drucken" abstellen, wie z.B. Kursivdruck der Pascal-Kommentare u.ä. In das Eingabefeld "Kommandozeilen-Parameter" steht standardmäßig wahrscheinlich "$NOSWAP /EPSON". Der erste Teil "$NOSWAP" muß stehenbleiben, der Eintrag "/EPSON" ist druckerspezifisch für Epson-Drucker. Bei Nicht-Epson-Druckern ist der Eintrag zu ändern in "/HP" für Drucker von HP (Hewlett & Packard) oder HP-kompatible Drucker mit der Druckersprache PCL (Page Control Language) oder in "/PS" für PostScript-Drucker. Es ist sinnvoll die CheckBox "Syntaxhervorhebung" zu aktivieren. Damit werden reservierte Wörter fettgedruckt, und – je nach den Einstellungen in "PRNFLTR.PAS" – noch einiger Schnickschnack mehr.

DOS aufrufen Damit kann die IDE temporär verlassen werden. Es erscheint der haßgeliebte DOS-Bildschirm mit der DOS-Eingabezeile und dem aktivierten Kommandointerpreter, erkenntlich am DOS-Prompt. Man kann nun alles tun, was MS-DOS bietet, z.B. eine Diskette formatieren, Dateien löschen, andere Programme (EXE-, COM- und BAT-Dateien) aufrufen usw. Ruft man ein fehlerhaftes Programm auf, das zum Absturz führt, hat man keine Möglichkeit mehr, zur IDE zurückzukehren. Deshalb vor dem DOS-Aufruf mit "Alles speichern" alle Pascal-Dateien speichern**. Wichtig: Nach dem Ende der Aktivitäten auf DOS-Ebene kehrt man mit dem DOS-Befehl "EXIT" zur IDE zurück**.

Beenden Tastenkürzel "Alt+X". Beenden der Pascal-Sitzung. Wenn das Editfenster noch nicht gespeichert sind, erscheint eine Sicherheitsabfrage.

Nach den Untermenüpunkten werden in einer Liste bereits bearbeitete Pascal-Quelltexte zur Schnellauswahl mit Doppelklick zum Öffnen angeboten, siehe Abbildung 5-B03.

5.3 Der Hauptmenüpunkt Bearbeiten

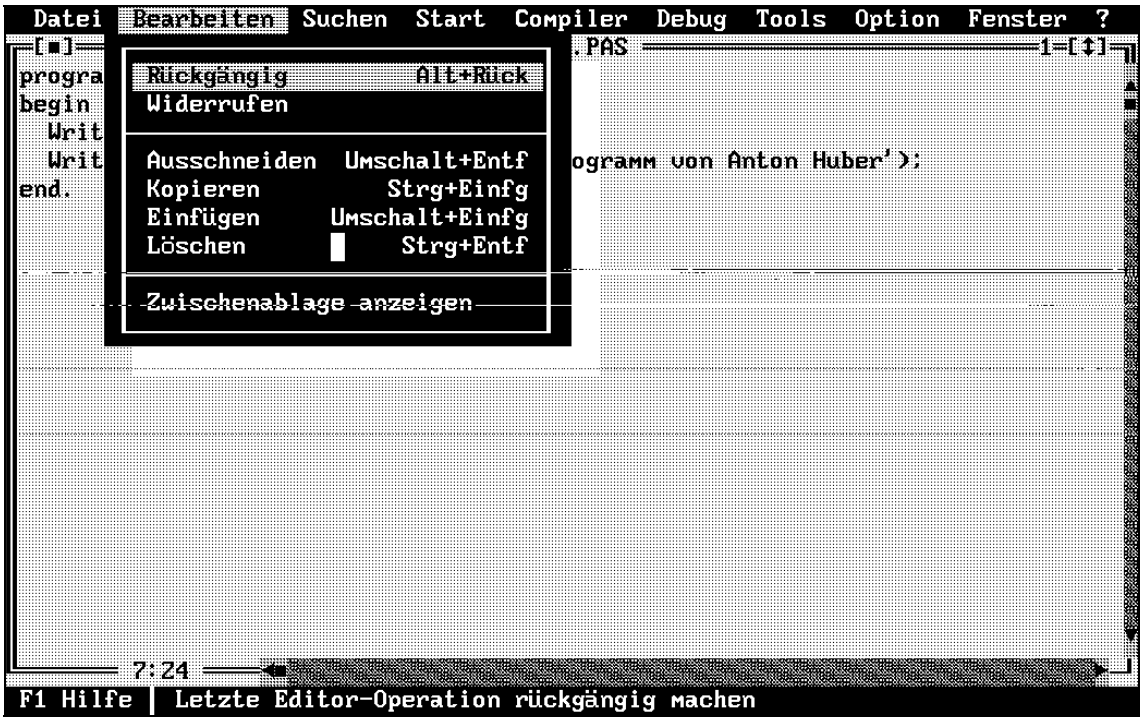

Abbildung 5-B04: Der Hauptmenüpunkt *Bearbeiten*

Der Hauptmenüpunkt *Bearbeiten* enthält folgende Untermenüpunkte (mit den unterstrichenen Zeichen können weitere Alt-Tastenkürzel gebildet werden):

Rückgängig Tastenkürzel **Alt+Rück**. Die letzte Editoperation kann rückgängig gemacht werden. Feinheiten: Wenn im Untermenüpunkt "Option/Umgebung/Editor..." die CheckBox "Befehlsgruppen widerrufen" akiviert ist, dann können Serien von Befehlen des gleichen Typs rückgängig gemacht werden.

Widerrufen Die Aktion "Rückgängig" kann widerrufen werden.

Ausschneiden Tastenkürzel **Umsch+Entf**. Der markierte Teil des aktuellen Editfenster wird in die Zwischenablage kopiert und aus dem Editfenster gelöscht. Die Zwischenablage ist wie bei Windows eine temporärer Speicher mit nur einer Speicherebene, d.h. es kann nur *ein* Objekt in der Zwischenablage stehen. Wird nochmals in die Zwischenablage kopiert, ist das alte Objekt überschrieben, egal ob das alte Objekt der komplette Quelltext eines großen Pascal-Programms waren und das neue nur ein Zeichen. Die Zwischenablage kann mit "Einfügen" wieder eingefügt werden, siehe dort.

Zum Markieren (Vorgriff Kap. 06, Editor): Vier Methoden stehen zur Wahl:

1. Mit gedrückter Maustaste ziehen.
2. Cursor auf das erste Zeichen setzen, dann Cursor bei gedrückter Taste "Umsch" mit beliebigen Cursortasten versetzen.
3. Das erste Zeichen mit Maus anklicken, dann Mauscursor auf das nächste Stelle nach dem gewünschten letzten Zeichen setzen und dann Maustaste und Taste "Umsch" gleichzeitig drücken.
4. Beim ersten Zeichen **Strg+K B** drücken (bei gedrückter Taste Strg die Taste K drücken und dann die Taste "B", die Taste "Strg" kann, muß aber nicht losgelassen werden) und

beim Zeichen nach dem letzten Zeichen **Strg+K K** drücken. Die Markierung wird am einfachsten durch einen Mausklick an einer nichtmarkierten Stelle aufgehoben oder mit **Stg+K H**.

Kopieren Tastenkürzel **Strg+Einfg**. Der markierte Teil des aktuellen Editfensters wird in die Zwischenablage kopiert (siehe "Ausschneiden") aber nicht aus dem aktuellen Editfensters gelöscht.

Einfügen Tastenkürzel **Umsch+Einfg**. Die Zwischenablage wird an der momentanen Cursorstelle im aktuellen Editfenster eingefügt.

Löschen Tastenkürzel **Strg+Entf**. Der markierte Teil des Editfensters wird gelöscht. Der Vorgang kann erfreulicherweise mit "Rückgängig" wieder rückgängig gemacht werden. Wer hat nicht schon versehentlich wichtige Programmteile gelöscht?

Zwischenablage anzeigen Die Daten in der Zwischenablage werden in einem eigenen Fenster angezeigt. Mitunter ganz praktisch, wenn man nicht mehr weis, was man vor 2 Stunden in die Zwischenablage kopiert hat.

5.4 Der Hauptmenüpunkt Suchen

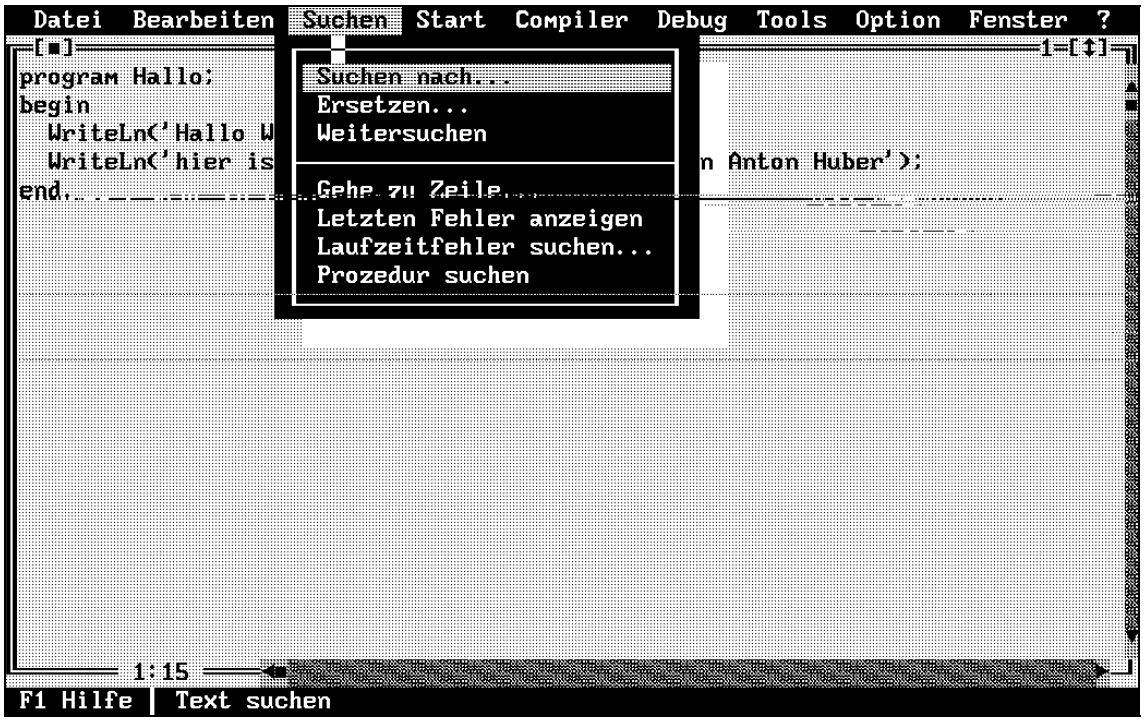

Abbildung 5-B05: Der Hauptmenüpunkt Suchen

Der Hauptmenüpunkt Suchen enthält folgende Untermenüpunkte (mit den unterstrichenen Zeichen können weitere Alt-Tastenkürzel gebildet werden):

Suchen nach... Suchen im aktuellen Editfenster nach gewünschten Begriffen. Es erscheint das (selbsterklärende) Dialogfenster "Text suchen", siehe spätere Abbildung 5-B06. Wenn der Cursor innerhalb eines Wortes steht, wird dieses Wort standardmäßig in das Eingabefeld "Suchen nach" kopiert.

Ersetzen.... Suchen und Ersetzen im aktuellen Editfenster. Es erscheint das (selbsterklärende) Dialogfenster "Text ersetzen", siehe Abbildung 5-B07. Man beachte die CheckBox "Mit Bestätigung" und die OptionButtons "Gesamter Text" und "Markierter Text".

Weitersuchen Letztes Suchen oder Ersetzen wiederholen.

Gehe zu Zeile... Es erscheint ein Dialogfenster, in dem die Nummer der Zeile eingegeben werden kann, zu der der Cursor im Editfenster versetzt werden soll.

Letzten Fehler anzeigen Cursor im Editfenster auf die Stelle des letzten Compilerfehlers versetzen. Zugleich wird eine Fehlernummer und eine Fehlermeldung angezeigt.

Laufzeitfehler suchen... Nicht für die ersten Anfängerschritte: Beim Ausführen eines compilierten Pascal-Programms (EXE-File oder TPU-File) kann es zu einem Laufzeitfehler kommen, z.B. Division durch Null, Lesen von einer nicht existierenden Datei, Ausgabe auf einen nicht bereiten Drucker u.ä. mehr. Das Programm bricht dann mit einer Laufzeitfehlermeldung ab, die lautet: "Runtime error *xx* at *ssss*:*oooo*". Für "xx" steht eine Fehlernummer, für "ssss:oooo" die Speicheradresse in der Intel-Uralt-Segment-Offset-Notation. Diese (hexadezimalen) Speicheradressen sind zu notieren. Wenn der zugehörige Pascal-Quelltext im Editfenster zur Verfügung steht, ruft man "Laufzeitfehler suchen..." auf und gibt in das dann erscheinende Dialogfenster die Speicheradresse in der gemeldeten Form ein. Nach dem "OK" wird das Programm bis zu dieser Speicheradresse abgearbeitet und somit der Fehler im Quelltext lokalisiert. Voraussetzung für die Laufzeitfehlersuche ist die Aktivierunng der Checkbox "Debug-Informationen" im Menü "Option/Compiler..." oder der Eintrag des Compilerbefehls "{$D+} im Quelltext, siehe Kap. 5.8.

Prozedur suchen Deklaration einer Prozedur oder Funktion suchen. Steht erst nach einer Compilation verfügbar. Der Cursor wird auf das zugehörige "begin" der Prozedur oder der Funktion gesetzt und nicht auf "procedure" bzw. "function".

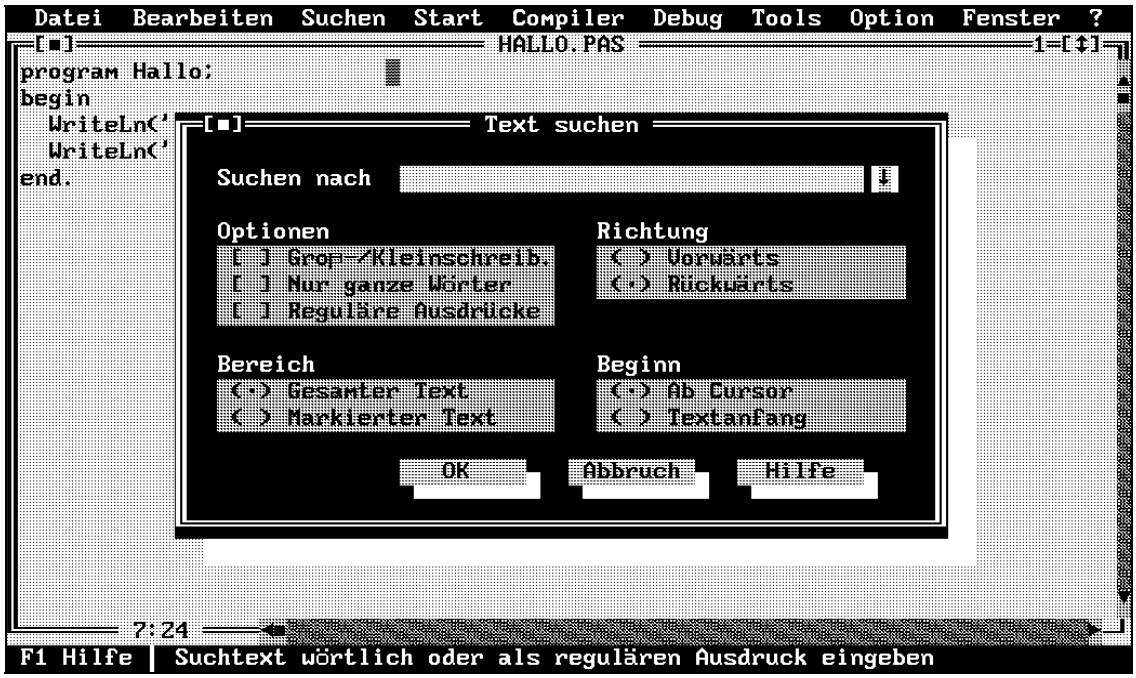

Abbildung 5-B06: Das Dialogfenster "Text suchen" des Menüpunktes "Suchen/Suchen nach..."

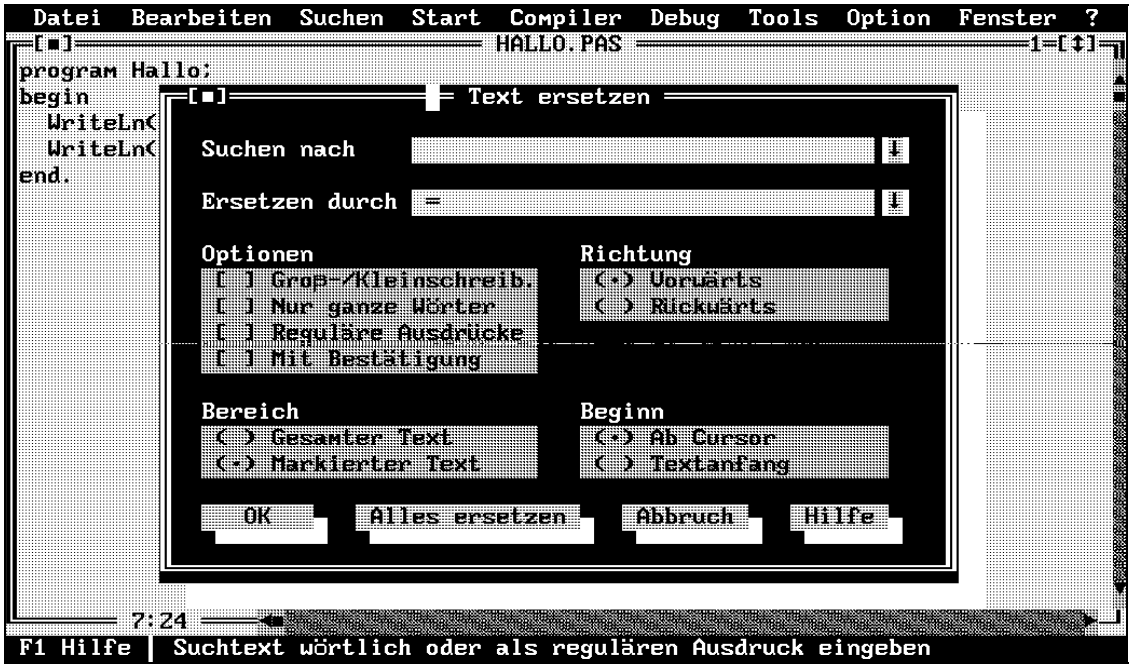

Abbildung 5-B07: Das Dialogfenster "Text ersetzen" des Menüpunktes "Suchen/Ersetzen..."

5.5 Der Hauptmenüpunkt Start

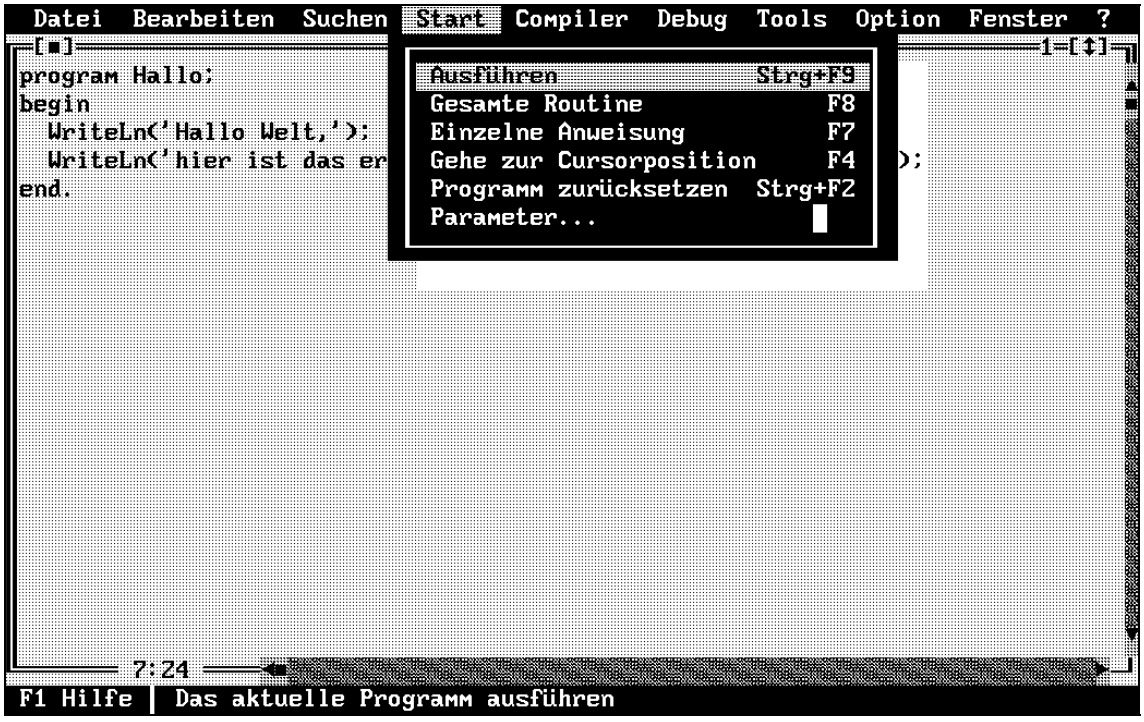

Abbildung 5-B08: Der Hauptmenüpunkt Start

Der Hauptmenüpunkt Start enthält folgende Untermenüpunkte (mit den unterstrichenen Zeichen können weitere Alt-Tastenkürzel gebildet werden):

Ausführen Tastenkürzel **Strg+F9**. **Für den Anfänger das wichtigste Tastenkürzel:** Compilieren *und* Ausführen des Programms. Im Fehlerfall wird das Compilieren abgebrochen und im Editfenster zu der betreffenden Stelle gesprungen und eine Fehlermeldung angezeigt. Wenn das Programm bereits compiliert ist, z.B. mit "Alt+F9", dann wird es mit "Strg+F9" nur ausgeführt.

Das Compilat wird bei Turbo-Pascal (nicht bei Borland Pascal für den Protected Mode) standardmäßig im Arbeitsspeicher abgelegt, d.h. es wird keine EXE-Datei erzeugt, es sei denn, daß mit dem (späteren Unter-) Menüpunkt "Compiler/Ausgabeziel" als Ausgabeziel nicht "Speicher" sondern "Festplatte" eingestellt wurde.

Gesamte Routine Tastenkürzel **F8**. Debug-Funktion. Durch Wiederholung von "F8" schrittweises Ausführen eines Programms zur Fehlerlokalisierung **ohne Anzeige der Einzelschritte von Routinen** (Prozeduren und Funktioenen). Siehe unbedingt "5.7 Hauptmenüpunkt Debug".

Einzelne Anweisung Tastenkürzel **F7**. Debug-Funktion: Wie vorher, jedoch **mit Anzeige der Einzelschritte von Routinen**. Siehe unbedingt "5.7 Hauptmenüpunkt Debug".

Gehe zur Cursorposition Tastenkürzel **F4**. Debug-Funktion: Programm wird bis zur Zeile vor dem Cursorstelle im Quelltext ausgeführt. Siehe unbedingt "5.7 Hauptmenüpunkt Debug".

Programm zurücksetzen Tastenkürzel **Strg+F2**. Debug-Funktion. Beenden des Debuggers und Zurücksetzen des Programms. Siehe unbedingt Kap. 5.7 "Hauptmenüpunkt Debug".

Parameter... Wird erst im Kap. 24 genauer behandelt und demonstriert. Nur für Neugierige: Nach Aufruf dieses Untermenüpunktes erscheint ein Dialogfenster mit der Bezeichnung "Kommandozeilenparamter", das im wesentlichen das Eingabefeld "Parameter" enthält. Mit einer entsprechenden Eingabe kann man DOS-Kommandozeilenparameter simulieren und das Programm in der IDE diesbezüglich testen ohne es zu einem EXE-File compilieren zu müssen, was ansonsten nötig wäre. Nicht unbedingt lebensnotwendig, aber in der genannten Situation sehr praktisch.

5.6 Der Hauptmenüpunkt Compiler

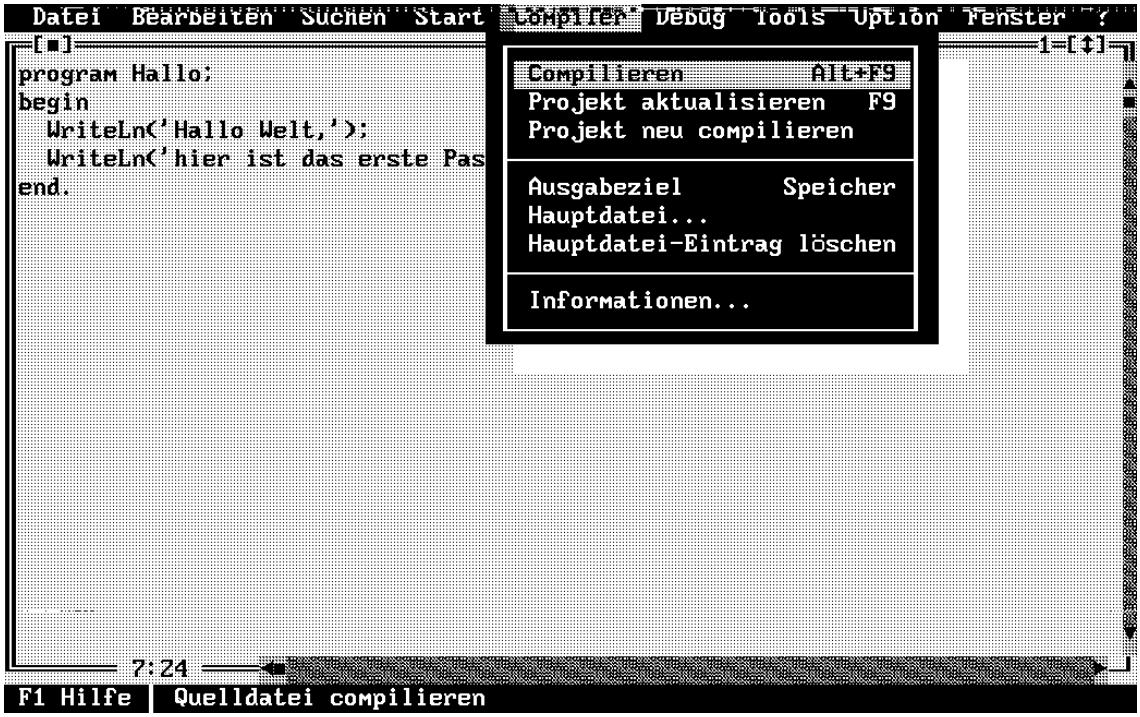

Abbildung 5-B09: Der Hauptmenüpunkt Compiler

Der Hauptmenüpunkt *Compiler* enthält folgende Untermenüpunkte (mit den unterstrichenen Zeichen können weitere Alt-Tastenkürzel gebildet werden):

Compilieren Tastenkürzel **Alt+F9**. Nur Compilieren des Programms, nicht ausführen. Für den Anfänger nicht von Belang, da mit **Strg+F9** (siehe früheren Punkt "Start/Ausführen") das Programm compiliert *und* ausgeführt wird. Nur notwendig, wenn bei größeren Projekten einzelne Units compiliert werden müssen. Im Fehlerfall wird das Compilieren abgebrochen und im Editfenster zu der betreffenden Stelle gesprungen und eine Fehlermeldung angezeigt.

Das Compilat wird bei Turbo-Pascal (nicht bei Borland Pascal für den Protected Mode) standardmäßig im Arbeitsspeicher abgelegt, d.h. es wird keine EXE-Datei erzeugt, es sei denn, daß mit dem (Unter-) Menüpunkt "Compiler/Ausgabeziel" als Ausgabeziel nicht "Speicher" sondern "Festplatte" eingestellt wurde.

Grundsätzliche Festlegung für die Ablage der EXE- und TPU-Dateien: Die EXE-Dateien und TPU-Dateien (compilierte Units) werden standardmäßig im aktuellen Verzeichnis angelegt, wenn nicht im Untermenüpunkt "Option/Verzeichnisse...", Eingabefeld "EXE-, TPU-Verzeichnis" ein Verzeichnis nach eigener Wahl eingetragen ist, siehe Kap. 5.9.

Projekt aktualisieren Tastenkürzel **F9**. Für den Anfänger nicht von Belang. Unabhängig von dem Eintrag in "Ausgabeziel" (siehe später) wird eine EXE-Datei oder eine TPU-Datei (Turbo-Pascal-Unit bei eigenen Units) erzeugt. Notwendig bei Projekten, die aus einer Hauptdatei und einer oder mehreren eigenen Units und/oder externen Assembler-Modulen bestehen. Der Name der EXE-Datei oder TPU-Datei wird aus dem Namen der Hauptdatei abgeleitet, sofern im Untermenüpunkt "Compiler/Hauptdatei..." ein entsprechender Eintrag vorliegt. Wenn nicht, dann wird der Dateiname der zuletzt in ein Editfenster geladenen Datei für die Namensvergabe der EXE-Datei oder TPU-Datei benutzt. Wenn sich der Quelltext von Units seit der

letzten Compilation geändert hat, dann werden diese (und nur diese) Units automatisch neu compiliert.

Projekt neu compilieren Wirkt ähnlich wie "Projekt aktualisieren", nur daß *alle* Komponenten des Projektes unabhängig von ihrem Aktualitätsstand *neu compiliert* werden. Compilation dauert somit u.U. etwas länger als bei "Projekt aktualisieren", sonst besteht kein Unterschied.

Ausgabeziel Bei Turbo-Pascal (arbeitet im DOS-Real-Mode) steht hier in der Regel "**Speicher**", was bedeutet, daß das das Compilat im Arbeitsspeicher abgelegt wird und somit beim Beenden der Pascal-Sitzung nicht mehr existiert (ausgenommen sind Projekte, siehe Unterpunkt vorher). Durch Einfachesklick auf "Ausgabeziel" ändert sich der Eintrag in "**Festplatte**". Beim nächsten Compilieren (mit "Strg+F9"oder "Alt+F9") wird dann das Compilat als EXE-Datei auf der Festplatte angelegt. Der Vorgang ist umkehrbar: Beim nächsten Anklicken von "Ausgabeziel" ändert sich der Eintrag wieder in "Speicher".
Anmerkung: Bei Borland-Pascal für den DOS-Protected-Mode (Aufruf mit "BP" statt "Turbo" wird das Compilat grundsätzlich immer auf der Festplatte als EXE- bzw. BPU-Datei abgelegt. BPU-Dateien (Borland Pascal Units) sind compilierte Units in Borland Pascal für den Protected-Mode.

Hauptdatei... Für den Anfänger nicht von Belang. Nützlich bei "Projekt aktualisieren" oder "Projekt neu compilieren". Wenn im Eingabefeld "Dateiname" des dann erscheinenden Dialogfensters "Hauptdatei" ein Eintrag steht (Name eines Pascal-Quelltextes) oder eingetragen wird, dann wird diese Datei compiliert, unabhängig von den Dateien in den Editfenstern. In einem Listenfeld werden weitere Pascal-Dateien (soweit vorhanden) aus dem aktuellen Verzeichnis zur Bestimmung als Hauptdatei angeboten. Doppelklick genügt. Im Dialogfenster "Hauptdatei" kann der Hauptdatei-Eintrag (nicht die Datei selbst) auch wieder gelöscht werden.

Hauptdatei-Eintrag löschen Der Hauptdatei-Eintrag wird ohne weitere Anzeige oder Abfrage gelöscht. Kann besser unter visueller Kontrolle im Dialogfenster bei "Hauptdatei..." bewerkstelligt werden. Dieser Untermenüpunkt ist somit überflüssig wie ein Kropf!

Informationen... Für Fortgeschrittene und Neugierige: In einem Fenster werden Information zum **Programm** angezeigt und zwar: Anzahl der compilierten Zeilen, Codegröße, Datengröße, Stackgröße, minimale und maximale Heap-Größe, Status ob bereits compiliert oder nicht, DOS-Speicherbelegung (DOS, IDE, Symbole, Programm, frei) und EMS-Speicherbelegung (IDE, andere, frei).

5.7 Der Hauptmenüpunkt Debug

Vorbemerkungen zum Debugger

Ein *Debugger* ist ein Hilfsprogramm zur Fehlersuche und -beseitigung in einem Programm. Der Computer-Sage nach stammt das Wort von einer toten Wanze (engl. bug), die in einem Relaiskontakt eines Steinzeit-Computers in den 40er Jahren für rätselhaftes Verhalten des Programms verantwortlich war. Debuggen heißt also ein Programm entwanzen oder von Fehlern befreien.

Das Debuggen kann auf Maschinensprach-Ebene oder auf Quelltext-Ebene erfolgen. Man unterscheidet:

- **Maschinensprache-Debugger**. Sie ermöglichen schrittweises Ausführen von Prozessorbefehlen und die Ausgabe der Inhalte von Prozessor-Registern und von Speicherzellen und auch deren Ver-

Veränderung. Wenn kein Quelltext und keine Hilfsdateien zur Verfügung stehen, ist man auf die spröden Maschinensprache-Debugger angewiesen. Ein Beispiel dafür ist der im Betriebssystem **MS-DOS** enthaltene Debugger mit dem Namen **DEBUG.EXE**. Dieser Debugger bietet darüber hinaus aber auch noch die Möglichkeit zum Disassemblieren eines Maschinenprogramms (Umwandlung der Prozessorbefehle in mnemo-technische Bezeichnungen) und kann zudem auch als einfacher Assembler (Umwandlung der mnemo-technischen Bezeichnungen in Prozessorbefehle) und zum Erstellen eines kurzen Maschinenprogramms verwendet werden.

- **Symbolische Debugger**. Sie arbeiten ebenfalls auf Maschinensprache-Ebene, können aber auf die Symbole des Programms (Namen von Variablen, Prozeduren und Funktionen usw.) zugreifen, wenn beim Linken des Programms eine Hilfsdatei (MAP-Datei) angelegt wird. Ein Beispiel dafür ist der Borland *Turbo-Debugger*, der zusammen mit dem *Turbo-Assembler* angeboten wird.
- **Quelltext-Debugger**. Wie der Name besagt, arbeitet man bei diesem Debugger auf der Ebene des Quelltextes und braucht sich somit nicht um die hardware-spezifischen Dinge wie Prozessor-Register und Speicher kümmern. Voraussetzung ist natürlich die Verfügbarkeit des Quelltextes, was bei der Programmentwicklung naturgemäß gegeben ist.

Borland-Pascal verfügt über zwei Debugger, die eine Mischung aus symbolischen Debugger und Quelltext-Debugger darstellen:

- einen in der IDE intergrierten Debugger
- einen externer Debugger.

Der externe Debugger (sehr leistungsfähig, umfangreich und eigenes Handbuch mit ca. 400 Seiten) wird für "Borland Pascal für den Protected-Mode", Programm "BP.EXE", zwingend gebraucht. Er kann aber auch für "Borland-Pascal für den Real-Mode", (Programm "Turbo.EXE", das "normale" Turbo-Pascal) eingesetzt werden. In der Grundausbildung wird vorrangig Turbo-Pascal eingesetzt; in diesem Fall ist der integrierte Debugger praktischer. Selbst dieser ist ziemlich umfangreich und kann zumindest vom Anfänger nicht voll ausgereizt werden.

Die Wahl zwischen den beiden Debuggern wird in der IDE getroffen, und zwar über die Menüpunkte "Option/Debugger...". Im dann erscheinenden Dialogfenster "Debugger-Optionen" werden beiden Checkboxen

[ ] Integrierter Debugger
[ ] Externer Debugger

angeboten. Es ist "integrierter Debugger" anzuwählen und bei Bedarf zusätzlich auch der "Externe Debugger".

Der in der IDE integrierte Debugger von Turbo-Pascal

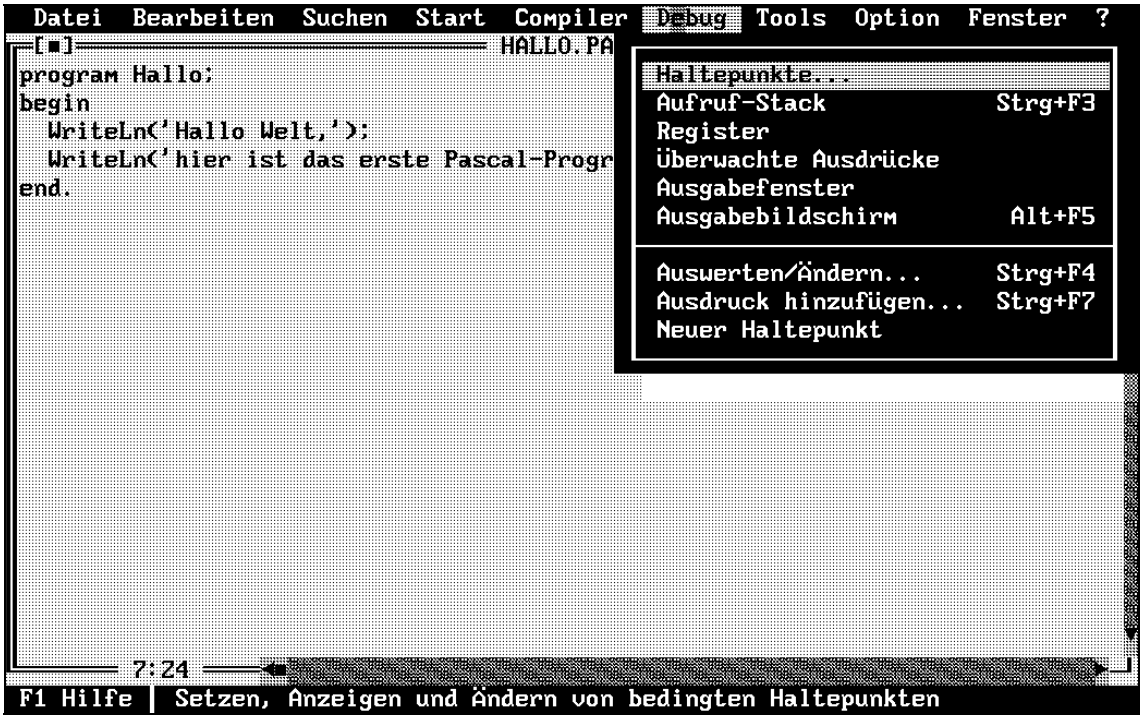

Abbildung 5-B10: Der Hauptmenüpunkt Debug

Folgende vier Debug-Funktionen sind bereits im Menüpunkt **Start** enthalten, werden aber hier detaillierter beschrieben:

Gesamte Routine **F8** Durch Wiederholung von **F8** schrittweises Ausführen eines Programms (Tracing, zur Fehlerlokalisierung) **ohne Anzeige der Einzelschritte von Routinen** (Prozeduren und Funktionen, soweit welche vorliegen). Die Haltezeilen, ab jetzt Haltepunkte genannt, werden in der Farbe cyan hervorgehoben. Die hervorgehobene Zeile ist aber nicht mehr ausgeführt worden. An den Haltepunkten hat man die Möglichkeit, mit **Strg+F4** das Dialogfenster "Auswerten und Ändern" aufzurufen und z.B. den Wert von Variablen abzufragen, siehe später.

Einzelne Anweisung **F7** Wie vorher, jedoch **mit Anzeige der Einzelschritte** in Routinen.

Gehe zur Cursorposition **F4** Programm wird bis zur Zeile *vor* der Cursorstelle im Quelltext ausgeführt. Der Haltepunkt wird in der Farbe cyan hervorgehoben. Die hervorgehobene Zeile ist aber nicht mehr ausgeführt worden. An den Haltepunkten hat man die Möglichkeit, mit **Strg+F4** das Dialogfenster "Auswerten und Ändern" aufzurufen und z.B. den Wert von Variablen abzufragen, siehe später. Mit **F4** kann immer nur ein Haltepunkt gesetzt werden. Es kann aber jederzeit der Cursor versetzt werden, um dann mit nochmaligem F4 bis zum neuen Haltepunkt auszuführen.

Programm zurücksetzen **Strg+F2** Debugger beenden und Programm in den Anfangszustand zurückversetzen. Die cyan-farbigen Markierungen der Haltepunkte werden gelöscht. Evtl. offene Dateien werden geschlossen. Permanente Haltepunkte (siehe nächster Punkt) werden aber mit **Strg+F2** nicht aufgehoben.

Permanente Haltepunkte können in beliebiger Anzahl an der aktuellen Cursorstelle mit **Strg+F8** gesetzt und auch wieder gelöscht werden, da **Strg+F8** als Toggle wirkt. **Permanente Haltepunkte** werden mit **rotem Balken** dargestellt, können aber mit **Strg+F2** nicht gelöscht werden. Vor dem Abspeichern müssen permanente Haltepunkte unbedingt mit nochmaligem **Strg+F8** gelöscht werden, sonst werden diese im Quelltext mit abgespeichert. Nach dem Programmstart z.B. mit **Strg+F9**, bleibt das Programm an dieser Stelle stehen (die Farbe des Haltepunktes ändert sich vorübergehend in cyan). Man kann nun mit **Strg+F4** das Dialogfenster "Auswerten und Ändern" aufrufen, Variablen abfragen usw. Mit nochmaligen **Strg+F9** wird bis zum nächsten Haltepunkt ausgeführt. Das Spiel wiederholt sich.

Die wichtigste Debugger-Anwendung in Kürze

Der Debugger ist durch die relativ gute Benutzerführung intuitiv erlernbar. Die Beschreibung aller Funktionen ist aber frustrierend umfangreich. Da es in den meisten Fällen genügt, bei merkwürdigem Programmverhalten das Programm an einer bestimmten Stelle zu unterbrechen, um den Wert einer Variablen abzufragen, wird den Einsteigern für die Fehlersuche folgende Vorgehensweise vorgeschlagen:

- Im Quelltext die Zeile suchen, in der der Fehler vermutet wird (im Zweifelsfall beginne man ziemlich am Anfang), Den Cursor auf die Nachfolgerzeile setzen und das Programm mit der Funktionstaste **F4** starten. Das Programm bleibt dann *vor* der Zeile stehen, in der der Cursor steht. Die Cursorzeile wird durch einen cyanfarbigen Balken im Quelltext markiert (Haltepunkt); diese Zeile selbst ist aber nicht mehr ausgeführt worden.
- Wenn man gar keine Vermutung über die Fehlerstelle hat, dann das Programm mit **F8** oder **F7** schrittweise ausführen und dann:
- Mit **Strg+F4** das Dialogfenster "Auswerten und Ändern aufrufen" aufrufen, siehe Abbildung 5-B11. In das Eingabefeld "Ausdruck" schreibt man die fragliche Variable (im Beispiel ist es eine numerische Variable, sie kann aber von beliebigem Datentyp sein) und klickt dann auf die Schaltfläche "Auswerten" oder auch die Enter-Taste. Im Feld "Ergebnis" wird dann der momentane Wert der Variablen angezeigt.

Der gleiche Wert wird auch in das Eingabefeld "Neuer Wert" kopiert. Wenn man möchte, kann man in "Neuen Wert" einen anderen Wert eingeben, z.B. den man für richtig hält und "Ändern" anklicken. Bei der weiteren Ausführung des Programms wird dann die Variable mit dem geänderten Wert belegt.

Das Debuggen kann beliebig fortgesetzt werden. Dazu Cursor auf eine andere Zeile setzen und wieder **F4** und **Strg+F4** betätigen usw.

Das Dialogfenster "Auswerten und Ändern" kann auch aufgerufen werden, wenn man das Programm mit **F7** oder **F8** schrittweise ausführt. Siehe spätere Ausführungen.

- Die Debugger-Sitzung ist unbedingt mit **Strg+F2** zu beenden. Die Haltepunktmarkierungen werden wieder gelöscht und das Programm wird auf den Anfang zurückgesetzt.
- Erst nach einer gewissen Vertrautheit mit dieser einfachen Debugger-Anwendung wage man sich an den vollen Funktionsumfang.
- Anmerkung: Die Eingabefeldbezeichnung "Ausdruck" tut kund, daß man in dieses Feld nicht nur Variablen, sondern auch Konstanten und "echte Ausdrücke" eingeben kann, z.B. "x + 1". Man kann dieses Eingabefeld somit z.B. auch als **Taschenrechner** für Grundrechnungsarten und für die Umrechnung von Hex-Zahlen (Zeichen "$" vorrausssetzen) benutzen. Zulässige Operatoren: + - * / div mod shl shr or and xor. Zuässig sind weiter alle Standardfunktionen, die in der const-Deklaration verwendet werden dürfen. Das sind: Abs, Addr, Chr, Hi, IOResult, Length, Lo, MaxAvail, MemAvail, Ofs, Ord, Pred, Ptr, Round, Seg, SizeOf, Ptr, Sseg, Succ, Swap, Trunc.
- Das Eingabefenster "**Neuer Wert**" kann **nur** für **Variablen** (auch Elemente von Arrays oder Records, vorausgesetzt, sie sind nicht wiederum Arrays oder Records) genutzt werden, nicht aber für "echte Ausdrücke".

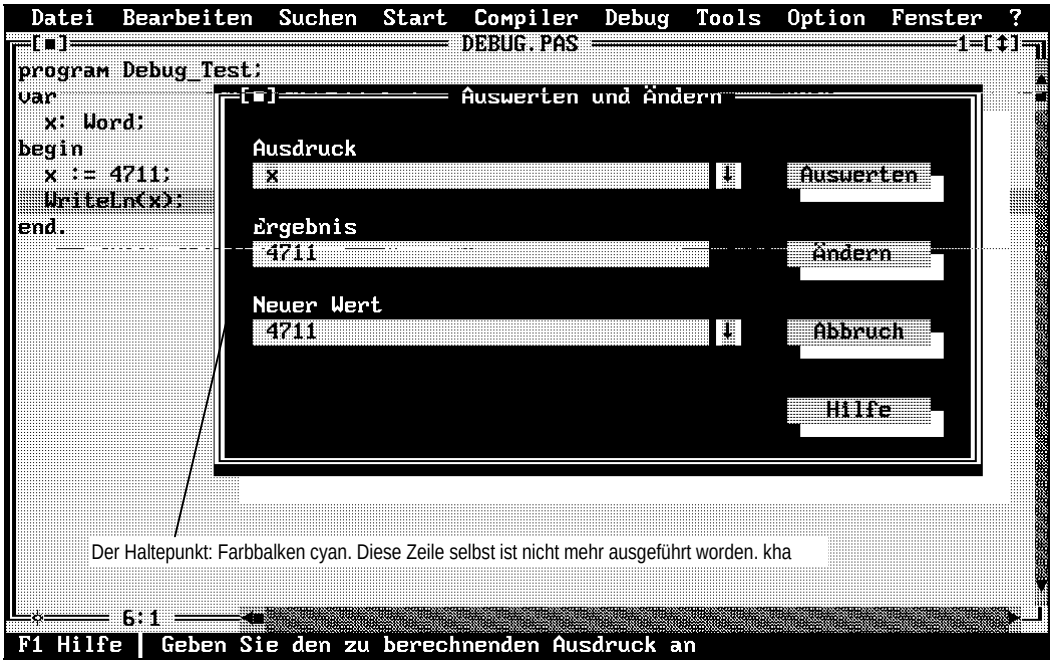

Abbildung 5-B11: Das Dialogfenster "Auswerten und Ändern" des Debuggers

Der Debugger in (ziemlich) vollständiger Darstellung

Der Hauptmenüpunkt *Debug* (Rückblende auf Abbildung 5-B10) enthält folgende Untermenüpunkte (mit den unterstrichenen Zeichen können weitere Alt-Tastenkürzel gebildet werden):

Haltepunkte... Leistungsfähigere Alternative zu **Strg+F8**. Es erscheint das Dialogfenster "Haltepunkte" in dem die bereits festgelegten permanenten Haltepunkte aufgelistet sind. Mit den Schaltfläche "Löschen" bzw. "Alle löschen" können permanente Haltepunkt gelöscht werden. Wenn man neue hinzufügen will, muß man (kurioserweise) die Schaltfläche "Ändern" anklicken". Es erscheint dann ein weiteres Dialogfenster "Haltepunkte ändern", in dem im Gegensatz zu **Strg+F8** aber auch Bedingungen für den Haltepunkt festgelegt werden können. In diesem Dialogfenster kann optional eine Bedingung, z.B. "`i > 11`" und/oder bei mehreren Durchläufen die Anzahl der Durchläufe festgelegt werden, bis der Haltepunkt tatsächlich ausgeführt wird. Zudem auch noch die Zeileennummer. Standardeinstellung ist die Zeile, in der der Cursor im Editfenster steht. Ganz schön, aber das hätte man etwas detaillierter ins Handbuch bzw. in die Online-Hilfe schrieben können.

Aufruf Stack Tastenkürzel **Strg+F3**. Bei einem Haltepunkt in einer Routine (Prozedur oder Funktion) werden die bis dahin durchlaufenen Routinen mit ihrem Namen angezeigt. Dazu zählt auch der Bezeichner nach "program" als Spezialfall einer Prozedur. Voraussetzung für den Aufruf ist die Aktivierung der Checkboxen "Debug-Informationen" und "Lokale Symbole" im Menü "Option/Compiler..." oder das Schreiben der entsprechenden Compilerbefehle "{$D+}" und "{$L+}" in den Quelltext. Siehe auch Kap. 5.9.

Register Zeigt die Inhalte der CPU-Register AX, BX, CD, DX, CS, DS, ES, SS, SI, DI und BP in hex an; das Flag-Register sinnvollerweise aber bitweise. Nur sinnvoll nach einem Haltepunkt. Beim nichtausgeführten Programm stehen in allen Registern nur Nullen.

Überwachte Ausdrücke Es werden in dem Fenster "Überwachte Ausdrücke" die im späteren Untermenüpunkt "Ausdrücke hinzufügen" definierten Ausdrücke (Watch-Ausdrücke) angezeigt. Dieser Unterpunkt ist zuerst zu lesen. Zum Löschen einzelner Ausdrücke ist aber das Fenster "Überwachte Ausdrücke" zu benutzen: Man stellt den Leuchtbalken auf den zu löschenden Ausdruck und betätigt dann die Enter.Taste. Es erscheint das Fenster "Überwachten Ausdrück ändern", in dem man den angezeigten Ausdrück nicht nur ändern, sondern auch löschen kann.

Ausgabefenster Zeigt den (DOS-) **Ausgabebildschirm als Fenster** an damit man sehen kann, was das Pascal-Programm bei der Ausführung Tolles getan geleistet hat. Beim Klick auf das Schließfeld des Ausgabefensters oder auf eine Stelle im Editfenster wird der Ausgabebildschirm wieder geschlossen und das Editfenster in seiner ursprünglichen Größe angezeigt.
Anmerkung: Bekanntlich wird in der IDE noch dem Ausführen automatisch wieder zur IDE mit seinem Menüsystem zurückgeschaltet, der (DOS-) Ausgabebildschirm verschwindet also sofort. Besser als der Einsatz von Debug-Funktionen (die dem Anwender des compilierten Programms sowieso nicht zur Verfügung stehen) ist es, wenn man durch eigene Programm-Maßnahmen am Ende dafür sorgt, daß der Ausgabebildschirm stehen bleibt, bis der Benutzter das tatsächliche Beenden wünscht. Die Holzhacker-Methode besteht darin, in die vorletzte Zeile des Hauptprogramms ein "ReadLn" zu schreiben. Der Benutzer muß dann wissen, daß er zum Beenden noch die Enter-Taste betätigen muß. Besser ist es, dem Benutzer im Programm mitzuteilen, mit welcher Taste er das Programm beenden soll. Mit "ReadKey" kann z.B. auch die Esc-Taste für solche Zwecke auserkoren werden, dazu später mehr im Kap. 07.

Ausgabebildschirm Tastenkürzel **Alt+F5**. Zeigt den (DOS-) **Ausgabebildschirm als Vollbild** an. Beim Klick auf eine beliebige Stelle des Fensters oder mit nochmaligen **Alt+F5** kehrt man zur IDE zurück. Sonst wie Punkt vorher.

Auswerten/Ändern... Das Grundsätzliche ist im vorangestellten Kasten "Die wichtigste Debug-Anwendung in Kürze" nachzulesen.
Weitere Details zu der Ausgabeart von verschiedenen Datentypen im Feld "Ergebnis" des Dialogfensters "Auswerten und Ändern". Die Ausführungen gelten auch für die Untermenüpunkte "Überwachte Ausdrücke" und"Ausdruck hinzufügen...".
Alle Integer-Typen: Ausgabe normal, siehe früheres Beispiel
Alle Real-Typen: In Gleitkomma- oder in Fixkomma-Schreibweise. Die letzgenannte Schreibweise (setzt Formatierparameter voraus, siehe später.
Char: Bei Ordnungsnummern >= 32 in Hochkomma eingeschlossen, sonst im Format #*n* oder #*nn*, wobei *n* bzw. *nn* die Ordnungsnummer nach Ascii bedeuten.
Booelean: `TRUE` oder `FALSE` in Großschreibweise.
Aufzählungstyp: In Großschreibweise.
String: In Hochkommas eingeschlossen.
Array: In runde Klammern eingeschlossen, die Einzelelemente mit Komma getrennt. Die Darstellung der Einzelelemente richtet sich nach ihrem Grundtyp. Beispiel für einen Integer-Array mit drei Elementen: (10,-12, 47,11). Bei vielen Elementen muß im Feld "Ergebnis" bzw. "Neuer Wert" horizontal gescrollt werden. Bei mehrdimensionalen Arrays erhalten die Elemente jeder Dimension nochmals eine Klammerung. Beispiel für einen Integer-Array mit 2 Zeilen und 3 Spalten: `((5,3,9),(1,4,3))`
Set: In eckige Klammern eingeschlossene und durch Kommas getrennte Element. Die Darstellung der Einzelemente richtet sich nach dem Grundtyp. Wenn möglich, werden in der Ausgabe zur Verkürzung mit ".." Unterbereiche gebildet.
Zwei Beispiele: `[MAI,JUNI,JULI]`und `['a'..'z']`.
Record: In runde Klammern eingeschlossene und durch Komma getrennte Liste der Elemente. Die Darstellung der Elemente richtet sich nach ihrem Grundtyp.
File: In runden Klammern eingeschlossene Ausgabe des Status (möglich: `OPEN`, `CLOSED`, `INPUT`, `OUTPUT`) und - mit Komma getrennt - des Dateinamens.

Pointer: Ausgabe im Format "PTR(Segment,Offset)". Adressen in hex. Beispiel: `PTR($5F0D,$4E).`

Zusätzlich können zur Angabe von Variablen oder Ausdrücken im Eingabefeld "Ausdruck" des Dialogfensters "Auswerten und Ändern" sowie bei "Überwachten Ausdrücken" noch mit einem oder z.T. auch mit mehrereren **Formatierparameter** angegeben werden. Dazu ist der Variablen (oder dem Ausdruck) ein **Komma** und ein **Formatiersymbol** nach folgender Auflistung in beliebiger Groß-/Kleinschreibung nachzusetzen. Leerzeichen bei mehreren Formatierparametern sind optional.

C Für Char und Strings. Auch Zeichen mit Ordnungsnummern < 32 werden dann als Zeichen dargestellt. Beispiel: `Ch, C`

S Für Strings. Die Zeichen < 32 werden im Format #nn dargestellt. Standard für Strings. Somit nur sinnvoll in Kombination mit dem späteren Formatierparameter **M**.

D Für dezimale Darstellung von Integer-Elementen. Standard und deshalb nur in Kombination mit dem späteren Formatierparameter **M** sinnvoll.

H Für hexadezimale Darstellung von Integer-Elementen.

$ Wie **H.** Das "$" ist Pascal-like.

X Wie **H.**

F *n* Floating Point-Darstellung (Festkommadarstellung) von Real-Elementen mit ***n***-1 Nachkommastellen. Wert ***n*** von 2 bis 11. (Dumme Frage des Autors: Warum wird bei n der Dezimalpunkt mitgezählt? Im Handbuch ist davon nicht die Rede!). Im Bedarfsfall wird gerundet. Beispiel für die Ausgabe der Real-Variablen `y` mit 4 Nachkommastellen: `y, F 5`

P Ausgabe der Adresse einer Pointe-Variablen im Format *Segment*:*Offset*, hexadezimal.

R Ausgabe aller Feldbezeichner eines Rekords *und* seiner Werte. Der Feldbezeichner erscheint in der Ausgabe großgeschrieben, dann folgt ein Doppelpunkt und dann der Wert des Feldbezeichners. Als Trennzeichen zwischen den einzelnen Feldern wird das Semikolon verwendet.

[***n***]**M** Memory Dump (Auszug aus dem Arbeitsspeicher). Es werden byteweise die Inhalte von sovielen Speicherstellen ausgegeben, wie die betreffende Variable belegt, wenn die Option ***n*** nicht benutzt wird; ansonsten soviele Speicherstellen ***n*** ab der ersten. Das niederwertigste Byte wird zuerst aufgeführt. Standardmäßig werden diese Speicherinhalte in hex (aber ohne Vorsatzzeichen "H" oder "$") ausgegeben.

In der Kombination (Reihenfolge beliebig, mit und ohne Leerzeichen) mit den bereits genannten Formatparameter **C** (Zeichenausgabe), **D** (dezimale Ausgabe) oder **H** (hexadezimal, Standard) ist jede Darstellungsform möglich. Zur Veranschaulichung: Es existiere eine Integer-Variable j mit dem Wert 4711 = 18*256 + 103. Sie belegt zwei Byte. Das niederwertige Byte hat den Wert 103, das höherwertige Byte den Wert 18, beide Werte dezimal, in hex $67 und $12. Die Eingabe "j, M" führt zur Anzeige "67 12" und ist hexadezimal zu verstehen, auch wenn kein $-Zeichen angezeigt wird. Die Eingabe "j, M H" ist klarer und führt zur Anzeige "$67 $12". Die Eingabe "j, M D" zeigt schließlich die Inhalte der beiden Speicherstellen mit "103 18" dezimal an. Mit "j, M C" erhält man die Char-Darstellung 'g↕ ', die aber bei Integer-Werten etwas unsinnig, aber dennoch richtig ist. Nach Ascii ist das Zeichen #103 das "g" und das Zeichen #18 der senkrechte Doppelpfeil**. Für Fortgeschrittene zur Übung empfohlen**: Man erzeuge in einem Programm die String-Variable "s" und belege sie mit 'Anton Huber, FH München, 80323 München'. Der String ist aktuell 38 Zeichen lang. Mit dem Debugger, genauer mit dem Dialogfenster "Auswerten und Ändern" lasse man sich die Variable wie folgt anzeigen "s, C M". Es wird angezeigt: '&Anton Huber, FH München, 80323 München'. Was bedeutet das erste Zeichen "&" in der Anzeige?

Ausdruck hinzufügen... Tastenkürzel "Strg+F7". Es erscheint das Dialogfenster "Ausdruck überwachen". Im Eingabefeld "Ausdruck" kann man den gewünschten Ausdruck (Watch-Ausdruck) eingeben; in der Regel werden es aber Variablen sein, deren Werte man kontrollieren möchte. Nach einem Haltepunkt können mit dem weiter oben beschriebenen Untermenü-

punkt "Überwachte Ausdrücke" die Bezeichnungen der Ausdrücke und die berechneten Werte in einem eigenen Fenster angezeigt werden. Es können mehrere Ausdrücke überwacht werden. Das Überwachen von Ausdrücken ist besonders dann anschaulich, wenn man das Programm schrittweise ausführt. Zum Löschen von überwachten Ausdrücken siehe früheren Untermenüpunkt "Überwachte Ausdrücke".
Was die **Formatierung** betrifft, gelten die entsprechenden Ausführungen wie bei "**Auswerten und Ändern**" auch hier. Deshalb dort nachlesen.

Neuer Haltepunkt Es erscheint das Dialogfenster "Neuer Haltepunkt", das aber genauso aufgebaut und zu bedienen ist wie das Dialogfenster "Haltepunkte ändern" im Untermenüpunkt "Haltepunkte..."; siehe oben.

Genug der Debuggerei!

5.8 Der Hauptmenüpunkt Tools

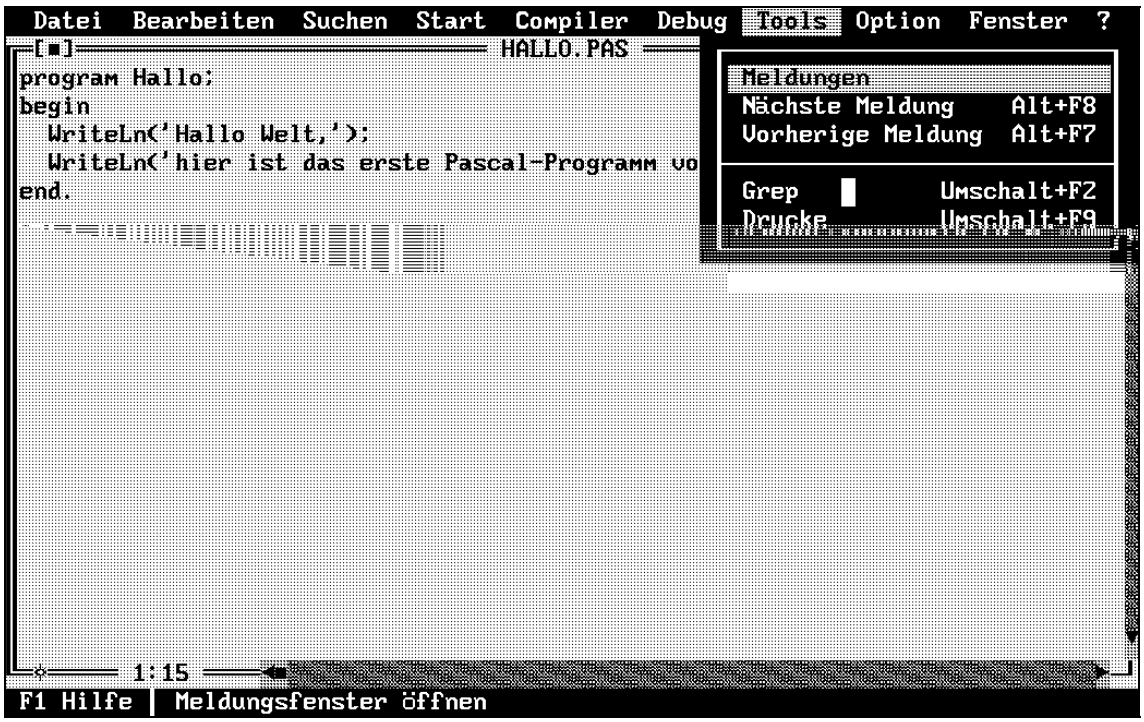

Abbildung 5-B12: Der Hauptmenüpunkt Tools

Der Hauptmenüpunkt *Tools* enthält folgende Untermenüpunkte (mit den unterstrichenen Zeichen können weitere Alt-Tastenkürzel gebildet werden):

Meldungen Zeigt in einem Fenster die Meldungen des Tools an.

Nächste Meldung Tastenkürzel **Alt+F8**. Der Menübalken wechselt zum nächsten Eintrag in der Liste.

Vorherige Meldung Tastenkürzel **Alt+F7**. Der Menübalken wechselt zum vorherigen Eintrag in der Liste.

Grep Tastenkürzel **Umsch+F2**. Hilfsprogramm (Tool) zur schnellen Durchsuchen von ein oder mehreren Dateien nach bestimmten Zeichenfolgen. Das Programm "Grep.COM", ca. 7 kByte,

besitzt viele Optionen und stammt aus der Unix-Welt, weshalb die Optionen mit "-..." und nicht DOS-like mit "/..." beginnen müssen. Das Programm steht bei Standardinstallation im Verzeichnis "C:\BP\Bin". Eine Übersicht über die Optionen erhält man, wenn man das Programm in der DOS-Eingabezeile mit "Grep ?" aufruft. Anmerkung: Im Norton Commander kann man mit "Alt+F7" auch relativ komfortabel suchen.
Es können auch noch weitere Hilfsprogramme installiert sein, im DR-Programmierlabor z.B. ist das Programm **Drucke** installiert (formatierter Ausdruck von beliebigen Textdateien, insbesondere für Pascal-Quelltexte). "Drucke" kann auch mit dem Tastenkürzel "Umsch+F9" aufgerufen werden. Zum Installieren eigener Hilfsprogramme dient das Untermenü "Option/-Tools...", siehe Kap. 5.9. Es können maximal 15 Tools installiert werden.

5.9 Der Hauptmenüpunkt Option

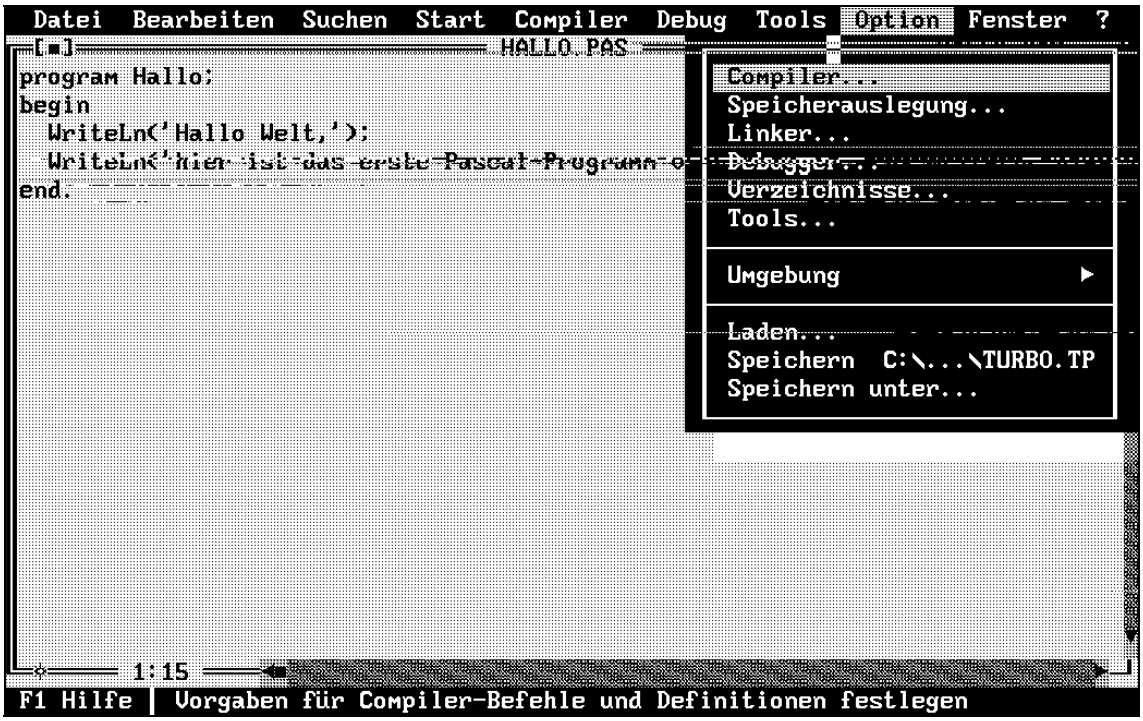

Abbildung 5-B13: Der Hauptmenüpunkt Option

Wie die Abbildung zeigt, enthält dieser Hauptmenüpunkt zehn Untermenüpunkte (mit den unterstrichenen Zeichen können in gewohnter Weise Tastenkürzel gebildet werden), die sich wiederum in weitere Untermenüpunkte aufteilen. Wegen des Umfangs werden die Abbildungen erst nach der folgenden Kurzbeschreibung und in dann unkommentierter Folge dargestellt. Die Kurzbeschreibung enthält nur die wichtigsten Informationen. Weitere Details können mit der Online-Hilfe erfahren werden.

Wichtig: Optionen dürfen auf den Rechnern im DR-Programmierlabor grundsätzlich nicht verändert werden; im Labor Computer Publishing nur nach Information des Laborleiters oder des Werkmeisters!

Compiler... Compiler-Befehle einstellen. Siehe Abbildung 5-B14. Das Dialogfenster enthält eine größere Anzahl von in Gruppen zusammengefaßten Checkboxen.

Vorab: Alle nachfolgenden **Compileroptionen** besitzen Äquivalente in Form von Compilerbefehlen. **Compilerbefehle** (Format "**{$..}**") können direkt in den Pascal-Quelltext geschrieben werden, was in manchen Fällen auch sinnvoller ist. Compilerbefehle haben höhere Priorität als die Einstellungen in "Option/Compiler...". Befindet man sich im Editfenster, können mit dem Tastenkürzel "**Strg+O O**" **alle Compileroptionen** in Form von Compilerbefehlen vor den Anfang des Quelltextes ausgegeben und editiert werden, siehe Beispiel in Kap. 5.12. Änderungen bei den ausgegebenen Compilerbefehlen haben keine Auswirkung auf die Einstellungen in "Option/Compiler...". Die ausgegebenen und evtl. geänderten Compilerbefehle werden mit dem Programm abgespeichert. Compilerbefehle sind meistens *Compilerschalter* mit den zwei Zuständen "+" oder "-". Beispiel: Der Compilerbefehl "**{$I-}**" schaltet die automatische Ein-/Ausgabeprüfung (entspricht der nichtaktivierten Checkbox "I/O-Prüfung") aus, bei "**{$I+}**" wird sie wieder eingeschaltet (entspricht der aktivierte Checkbox "I/O-Prüfung"). Compilerbefehle können lokal oder global wirken, siehe Kap. 5.12.

Aktivierte Checkboxen entsprechen dem Plus-Modus, nichtaktivierte dem Minus-Modus des Compilerbefehls. Um Doppelangaben zu vermeiden, sind die Beschreibungen der Optionen nur im Kap. 5.12.1 aufgeführt.

Zur Checkbox-Gruppe "**Code-Erzeugung**": Bei heutigen Rechnern sollten die Checkboxen "Word-Datenausrichtung " und "286-Instruktionen erzeugen" aktiviert.

Checkbox "Far-Aufrufe erzeugen"	Siehe Compilerschalter **F** in Kap. 5.12.1
Checkbox "Overlays möglich"	Siehe Compilerschalter **O** in Kap. 5.12.1
Checkbox "Word-Datenausrichtung"	Siehe Compilerschalter **A** in Kap. 5.12.1
Checkbox "286-Instruktionen erzeugen"	Siehe Compilerschalter **G** in Kap. 5.12.1

Zur Checkbox-Gruppe "**Laufzeitfehler**": Hier sollten zumindest im Entwicklungsstadium alle vier Checkboxen aktiviert sein. Der Compiler erzeugt dann einen etwas längeren und vielleicht auch etwas langsameren Code; diese Nachteile stehen aber in keinem Verhältnis zu den sonst möglichen Fehlern.

Checkbox "Bereichsüberprüfung"	Siehe Compilerschalter **R** in Kap. 5.12.1
Checkbox "Stack-Prüfung"	Siehe Compilerschalter **S** in Kap. 5.12.1
Checkbox "I/O-Prüfung "	Siehe Compilerschalter **I** in Kap. 5.12.1
Checkbox "Überlaufprüfung "	Siehe Compilerschalter **Q** in Kap. 5.12.1

Zur Checkbox-Gruppe "**Debugger**": Es sollten unbedingt beide Checkboxen aktiviert sein, um auch in kniffligen Fällen den Fehler mit dem Debugger lokalisieren zu können.

Checkbox "Debug-Informationen "	Siehe Compilerschalter **D** in Kap. 5.12.1
Checkbox "Lokale Symbole "	Siehe Compilerschalter **L** in Kap. 5.12.1

Zur Checkbox-Gruppe "**Syntax-Optionen**": Mit Ausnahme der "Erweiterten Syntax" sollten alle anderen Checkboxen aktiviert sein.

Checkbox "Strenge Prüfung von VAR-Strings "	Siehe Compilerschalter **V** in Kap. 5.12.1
Checkbox "Boolesche Ausdrücke vollständig"	Siehe Compilerschalter **B** in Kap. 5.12.1
Checkbox "Erweiterte Syntax "	Siehe Compilerschalter **X** in Kap. 5.12.1
Checkbox "Typisierte @Operator "	Siehe Compilerschalter **T** in Kap. 5.12.1
Checkbox "Offene Array-Grenzen"	Siehe Compilerschalter **P** in Kap. 5.12.1

Zur Checkbox-Gruppe "**Gleitkommaberechnungen**". Es sollten die Checkboxen "**80x87-Code**" und "**Emulation**" aktiviert werden; dann ist das Programm auf allen Systemen lauffähig egal ob sie mit **mathematischen Coprozessor** ausgestattet sind oder nicht. Der EXE-File ist aber größer. Der Coprozessor beschleunigt Gleitkommberechnungen erheblich.

Checkbox "80x87-Code"	Siehe Compilerschalter **N** in Kap. 5.12.1
Checkbox "Emulation"	Siehe Compilerschalter **E** in Kap. 5.12.1

Das Eingabefeld "**Definition für bedingte Compilierung**" ist im Normalfall nicht von Interesse. Falls Bedarf besteht, sind die Eintragungen zweckmäßiger in Form einer Compilerbedingung direkt in den Quelltext zu schreiben, siehe Kap. 5.12.3.

Speicherauslegung... Speicher-Optionen (Äquivalent zum Compilerbefehl "{$M .., ..., ..}"). Siehe Abbildung 5-B15. Das Dialogfenster "Speicherauslegung" zeigt die Größe des Stack-Speichers (Standard 16384 Byte, max. 65520 Byte, min. 1024 Byte), die minimale Größe des Heap-Speichers (Standard 0 Byte) und die maximale Größe des Heap-Speichers (Standard 655360 Byte) an. Alle Werte können verändert werden. Siehe Kap. 11 (Prozeduren und Funktionen) und Kap. 19 (Zeiger). Für einfache Programme (nicht zu tiefe Verschachtelung von Routinen mit vielen lokalen Variablen und keine zu tiefen Rekursionen) genügt die in der Abbildung 5-B15 gezeigte Grundeinstellung. Bei Rekursion kann aber schon mal die Fehlermeldung kommen, daß der Stack nicht ausreicht, dann ist der Wert höher zu setzen. Es ist aber dann besser, statt "Option/Speicherauslegung..." den entsprechenden Compilerbefehl "{$M ....}" zu benutzen und die gewünschten Werte vor den Anfang des Programms zu schreiben. Beispiel: `"{$M 65520, 0, 65520}".` Der Stack wird im Beispiel auf den max. Wert von 65520 Byte eingestellt, die anderen Werte bleiben beim Standard. Die Angaben im Programm haben höhere Priorotät als die entsprechenden Option-Einstellungen.

Linker... Für den Profi! Siehe Abbildung 5-B16. Es kann optional eine Map-Datei angelegt werden, die Informationen zur Fehlersuche enthält (z.B. Speicheradressen der Variablen und Routinen, auch Standardroutinen). Voraussetzung ist die Aktivierung der Checkboxen "Debug-Informationen" und "Lokale Symbole" im Menü "Option/Compiler..." oder das Schreiben der Compilerbefehle "{$D+}" und "{$D+}" in den Quelltext. In einer Optionsgruppe kann aus vier Optionen ausgewählt werden, wie detailliert die Map-Datei angelegt werden soll (Standard ist "Aus", also keine Map-Datei), desweiteren kann festgelegt werden, ob die Map-Datei als "echte" Datei auf die Platte geschrieben werden soll (dazu wird der aktuelle Dateiname, aber mit der Extension".MAP" benutzt) oder ob die Map-Datei im Arbeitsspeicher abgelegt werden soll. Auf die Platte wird nur geschrieben, wenn bei der Compilation ebenfalls auf die Platte compiliert wird.

Debugger... Es erscheint das Dialogfenster "**Debugger-Optionen**". Siehe Abbildung 5-B17. In der Checkbox-Gruppe "**Debugger**" können die Checkboxen "**Integrierter Debugger**" und/oder "**Externer Debugger**" angewählt werden. Es soll auf jeden Fall der integrierter Debugger angewählt sein. Diejenigen, die Zeit und Lust haben, können zusätzlich auch den externen Debugger anwählen und sich dann mit dem Handbuch "Turbo Debugger" beschäftigen, das in der Version 7.0 ca. 400 Seiten umfaßt. Der Autor schlägt den Anfängern vor, statt dessen Pascal zu lernen. In der Optionsgruppe "**Anzeige umschalten**" hat man die Wahl zwischen den drei Optionen: **Nie**, **Automatisch** und **Immer**, die für den integrierten Debugger gelten und festlegen, ob bei Ausführung des Programms zwischen den Anzeigefenstern umgeschaltet werden soll. Bei **Nie** wird nicht umgeschaltet, bei **Automatisch** (Standardeinstellung) nur dann, wenn das Programm Bildschirmausgaben erzeugt oder wenn Routinen aufgerufen werden, auch wenn diese selbst keine Bildschirmausgaben erzeugen. Die Option **Immer** ist dann zu wählen, wenn das Programm möglicherweise die Editfenster oder Teile der IDE überschreibt.

Verzeichnisse... Im Dialogfenster "Verzeichnisse" erscheinen vier Eingabe-Listfelder, siehe Abbildung 5-B18.

Im Eingabe-Listenfeld "**EXE-/TPU-Verzeichnis**" kann das Verzeichnis eingetragen werden, in dem bei Compilation auf die Platte die EXE- bzw. bei Units die TPU-Dateien abgelegt werden sollen. Wenn nichts eingetragen wird (Standard), dann werden diese Dateien im aktuellen Verzeichnis abgelegt, es sei denn, daß beim Abspeichern des Quelltextes oder Öffnen explizit ein anderes Verzeichnis angegeben wurde. Dann ist dieses Verzeichnis das aktuelle Verzeichnis, in dem die genannten Dateien abgelegt werden. Für "normale" Anwendung sollte man nichts eintragen. Mit Rücksicht auf die Datensicherheit auf den Rechnern im DR-

Programmierlabor wurde aber eingetragen "C:\Student", d.h. EXE- und TPU-Dateien landen immer in diesem "vogelfreien" Verzeichnis, in dem die Studenten auch ihre Quelltexte ablegen dürfen und sonst nirgendwo.

Im Eingabe-Listenfeld "**Include-Verzeichnisse**" werden die Verzeichnisse angegeben (Standard ist kein Eintrag) in dem der Compiler nach den Include-Dateien suchen soll, die mit dem Compilerbefehl "**{$I** *dateiname*}" im Quelltext aufgeführt sind (man kann aber ggf. bei "*dateiname*" auch einen Zugriffspfad angeben und sich den allgemeinen wirkenden Eintrag sparen). Man kann mehrere Verzeichnisse angeben; sie sind dann mit einem Semikolon zu trennen. Sie werdann in der angegebenen Reihenfolge durchsucht. Include-Dateien haben wegen der praktischeren Units keine besondere Bedeutung mehr.

Im Eingabe-Listenfeld "**Unit-Verzeichnisse**" werden die Verzeichnisse eingetragen (bei der Standard-Installation ist es "C:\BP\Units"), in dem die Turbo-Pascal eigenen Units (z.B. CRT, PRINTER, GRAPH, STRINGS usw.) abgelegt sind. Auch hier könnte man mehrere Units angeben (gibt hier keinen praktischen Sinn); sie sind dann mit Semikolon zu trennen.

Im Eingabe-Listenfeld "**Objekt-Verzeichnisse**" werden die Verzeichnisse eingetragen (Standard ist kein Eintrag), in denen OBJ-Dateien (Assembler-Routinen von einem Assembler oder von einem anderen Compiler, Extension ".OBJ") abgelegt sind, die im Quelltext mit dem Compilerbefehl "**{$L** *objektdateiname*}" eingebunden (gelinkt) werden sollen. Auch hier kann man mehrere Verzeichnisse angeben; sie sind dann mit Semikolon zu trennen. Beim Compilieren wird zuerst im aktuellen Verzeichnis nach der OBJ-Datei gesucht und erst dann in den angegeben Verzeichnissen.

Tools... Tools (Hilfsprogramme, siehe auch Menüpunkt "Tools") installieren, siehe Abbildung 5-B19 und 5-B20. Es werden zunächst die installierten Tools angezeigt, siehe Abbildung 5-B19, dann können mit den Schaltflächen "Neu" oder "Ändern" (beide sind gleichwertig) neue Tools – auch eigene Programme – installiert oder installierte wieder gelöscht werden, siehe Abbildung 5-B-20. Bei Bedarf können auch noch Kommandozeilen-Argumente für das Tool eingetragen werden, siehe Tool "Grep". Außerdem können mit der Tilde "~" selbst Alt-Tastenkürzel definiert werden, siehe Abbildung. Und schließlich können noch freie Umsch-Funktionstasten-Tastenkürzel festgelegt werden.

Umgebung IDE-Umgebung konfigurieren, siehe Abbildung 5-B21bis 5-B26. Es erscheint ein Untermenü mit den Schaltflächen "Vorgaben...", "Editor", "Maus...", "Start..." und "Farben...". Damit kann man sich die IDE den eigenen Vorstellungen anpassen; die Standardeinstellung ist aber ganz gut und darf im DR-Programmierlabor nicht verändert werden.

Laden... Konfigurationsdatei laden. Es erscheint das Dialogfenster "Konfigurationsdatei laden", mit der Standardeinstellung "Turbo.TP". Die Option "Laden..." ist nur dann von Interesse, wenn man mit verschiedenen Konfigurationsdateien arbeiten möchte. Dafür besteht in der Grundausbildung kein Bedarf.

Speichern Alle Einstellungen unter dem Hauptmenüpunkt "Option" gelten nur für die aktuelle Pascal-Sitzung. Sollen sie permanent werden, dann sind die Optionen mit dem Untermenüpunkt "Speichern" in der Konfigurationsdatei "Turbo.TP" zu speichern. Diese Datei befindet sich bei der Standardinstallation im Verzeichnis "C:\BP\Bin".

Speicher unter... Konfigurationsdatei mit anderen Namen und/oder in einem anderen Verzeichnis speichern (man hüte sich davor!)

Es folgt die Abbildungsserie für die Untermenüpunkte des Menüpunktes "Option":

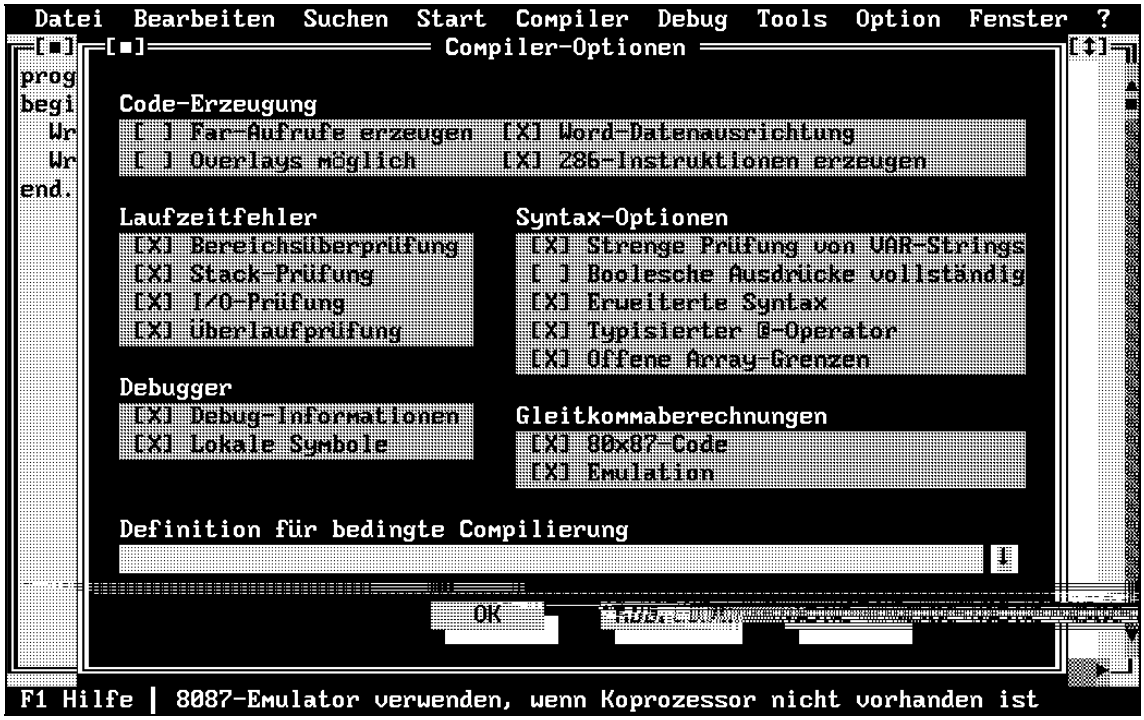

Abbildung 5-B14: Der Untermenüpunkt "Option/Compiler..."

Abbildung 5-B15: Das Dialogfeld "Speicherauslegung" des Untermenüpunktes "Option/Speicherauslegung..."

Abbildung 5-B16: Der Untermenüpunkt "Option/Linker..."

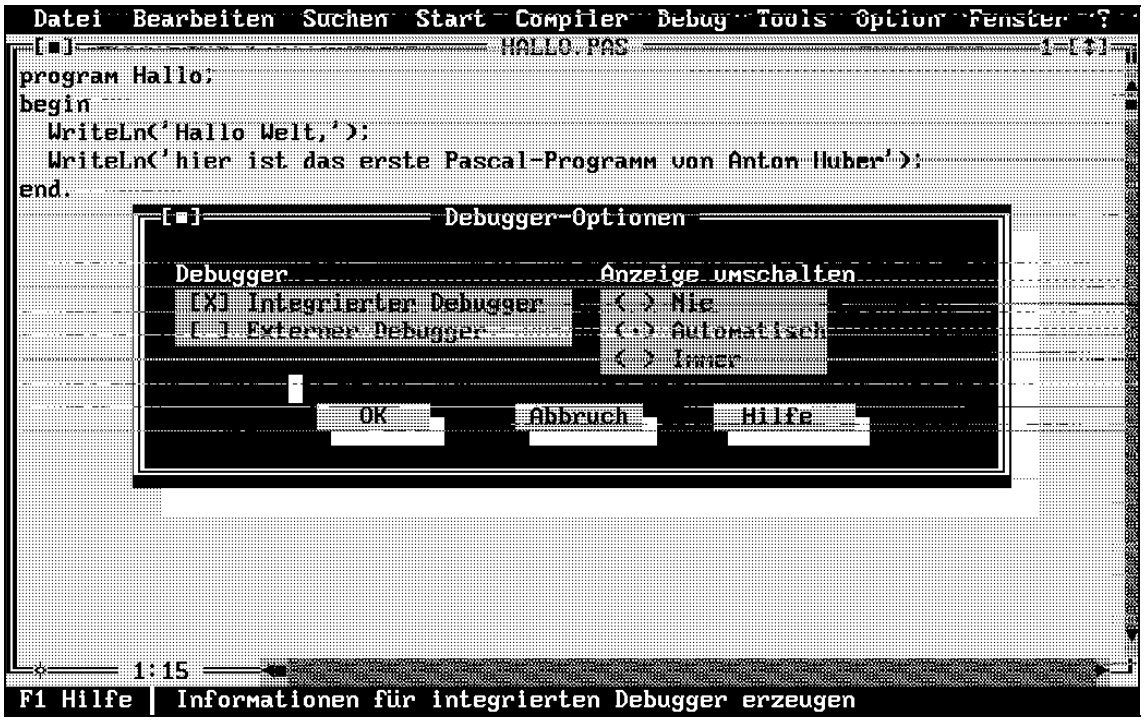

Abbildung 5-B17: Der Untermenüpunkt "Option/Debugger..."

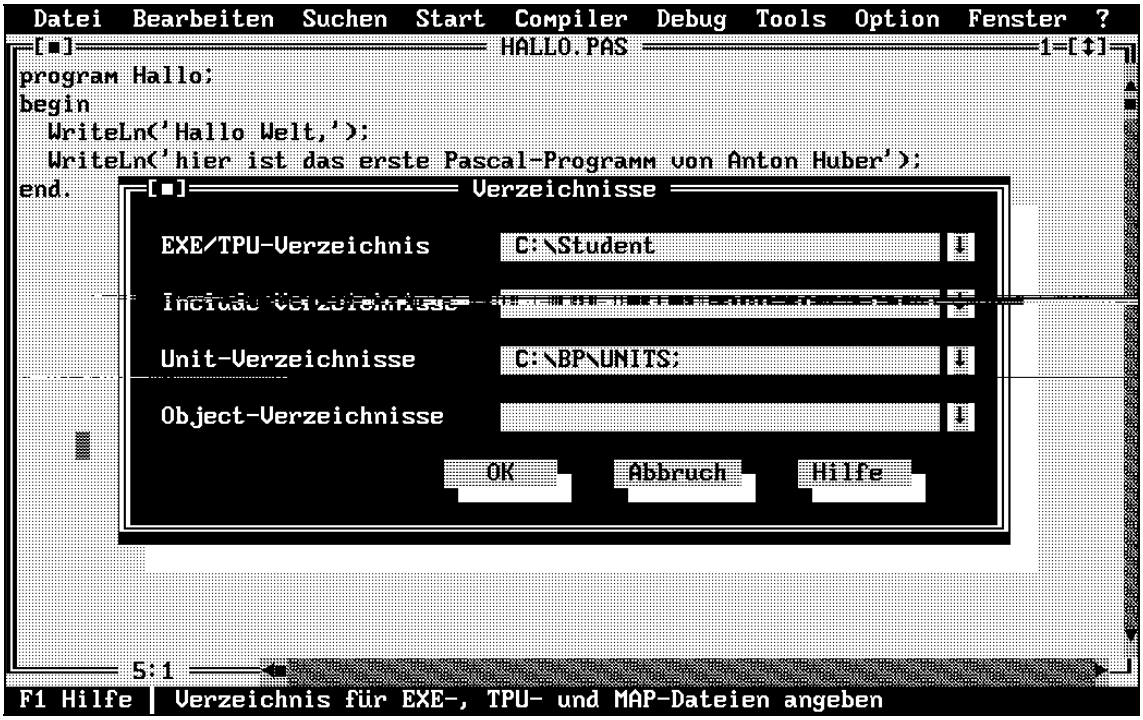

Abbildung 5-B18: Der Untermenüpunkt "Option/Verzeichnisse..."

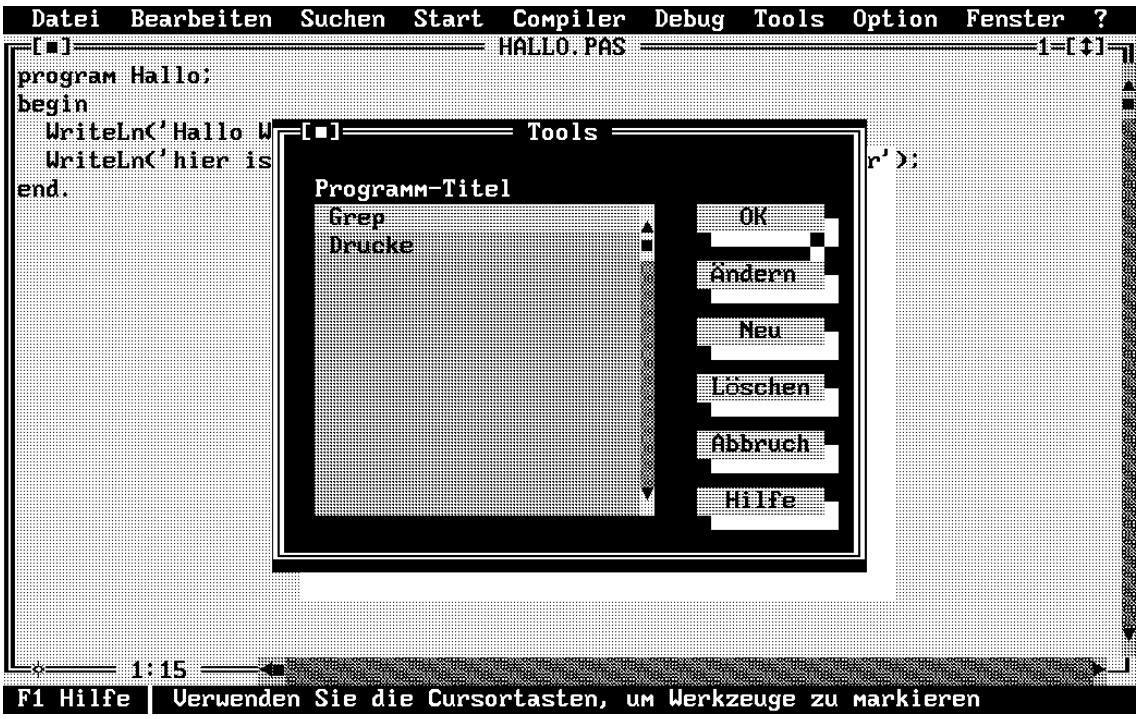

Abbildung 5-B19: Der Untermenüpunkt "Option/Tools..."

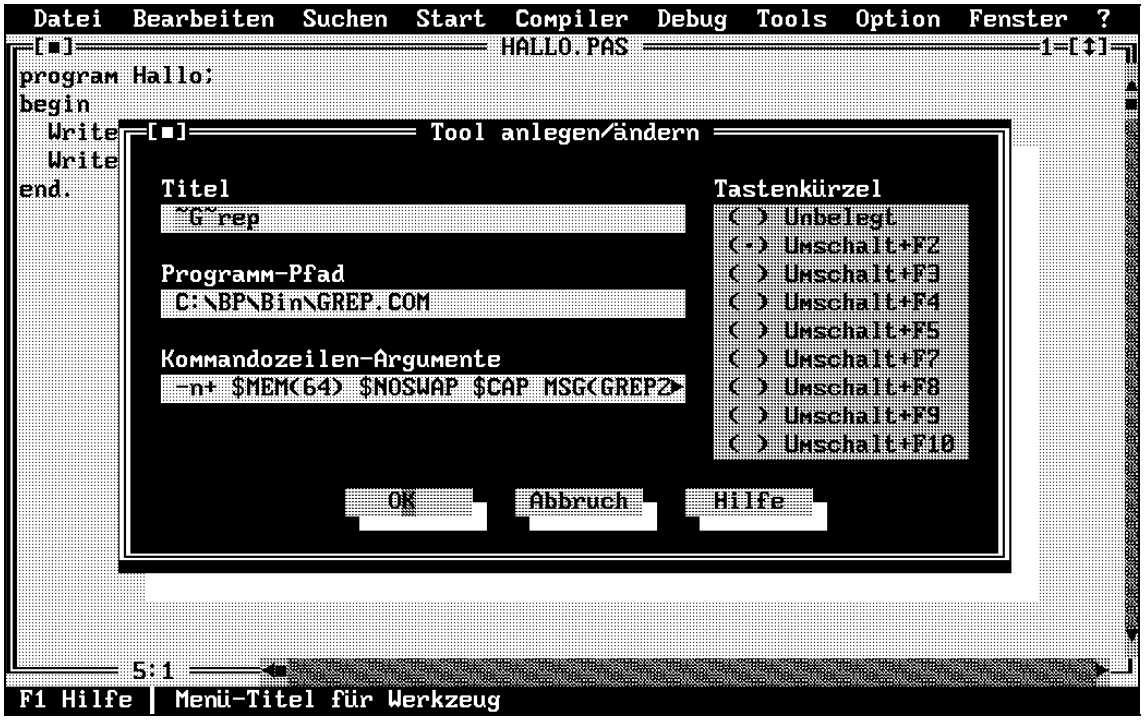

Abbildung 5-B20: Das Dialogfeld "Tool anlegen/ändern" des Untermenüpunktes "Option/Tools.../Neu" bzw. "Option/Tools.../Ändern"

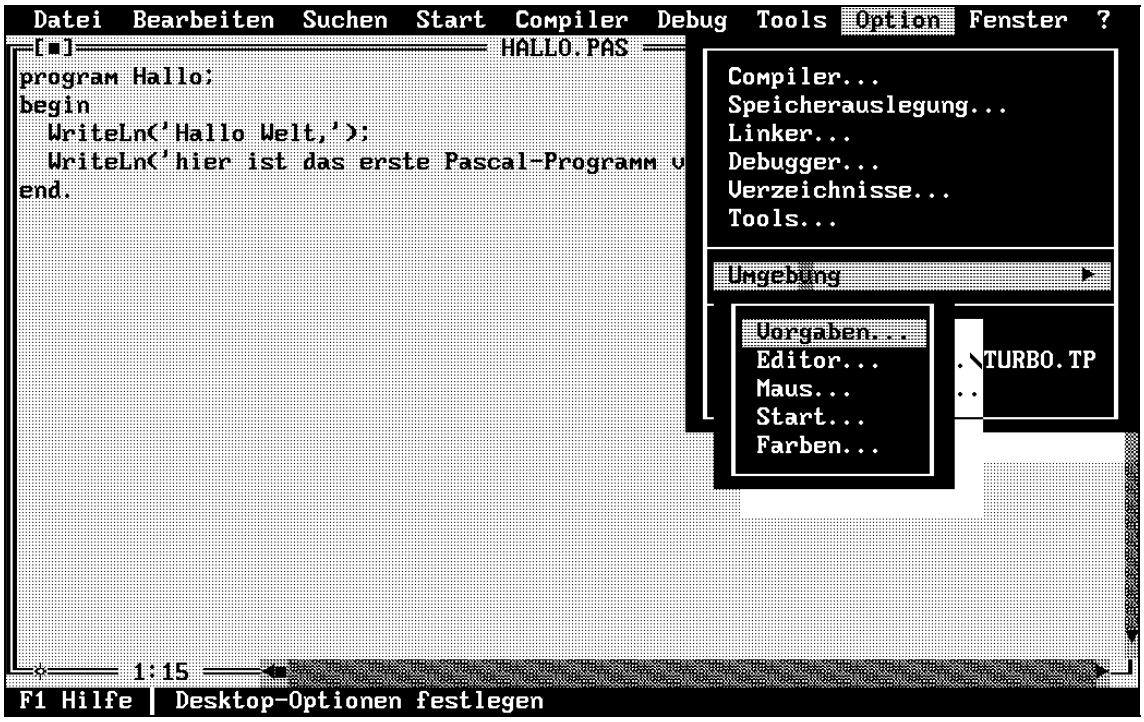

Abbildung 5-B21: Der Untermenüpunkt "Option/Umgebung"

Abbildung 5-B22: Der Untermenüpunkt "Option/Umgebung/Vorgaben..."

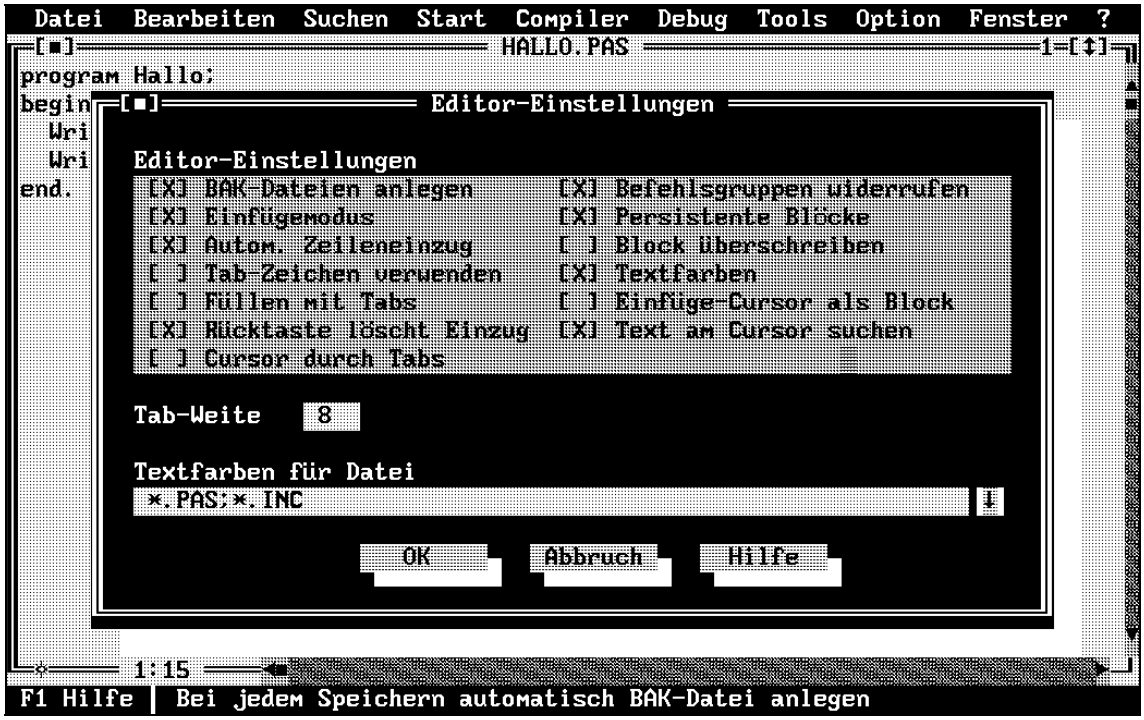

Abbildung 5-B23: Der Untermenüpunkt "Option/Umgebung/Editor..."

Abbildung 5-B24: Der Untermenüpunkt "Option/Umgebung/Maus..."

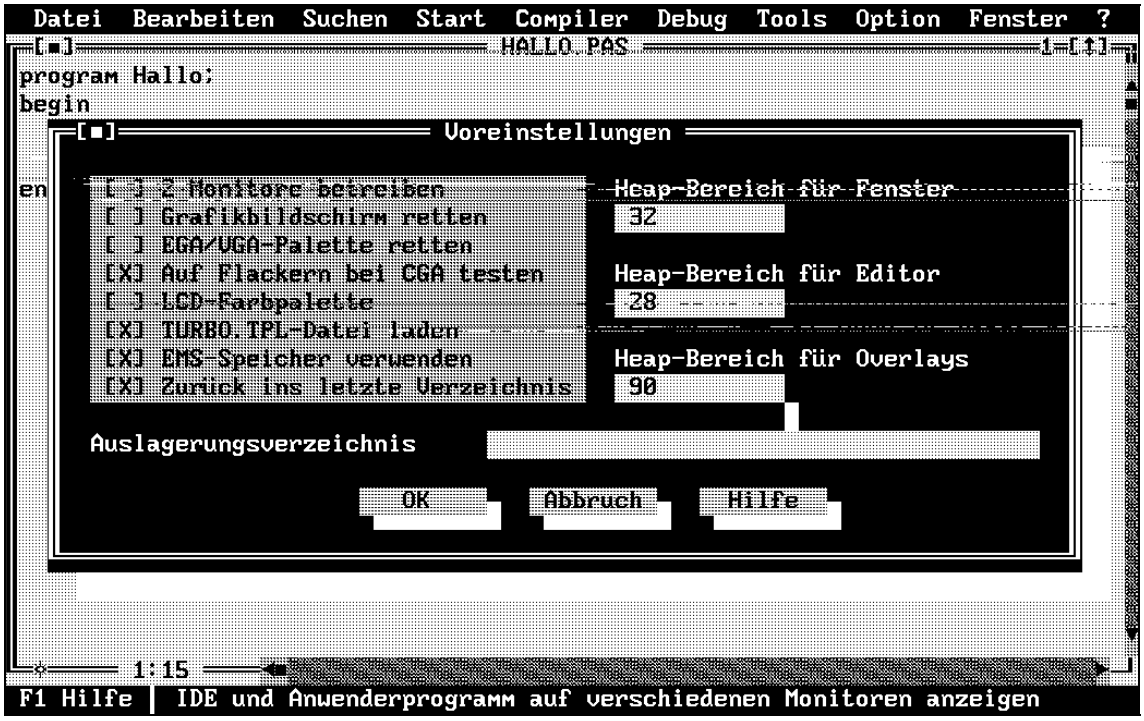

Abbildung 5-B25: Der Untermenüpunkt "Option/Umgebung/Start..."

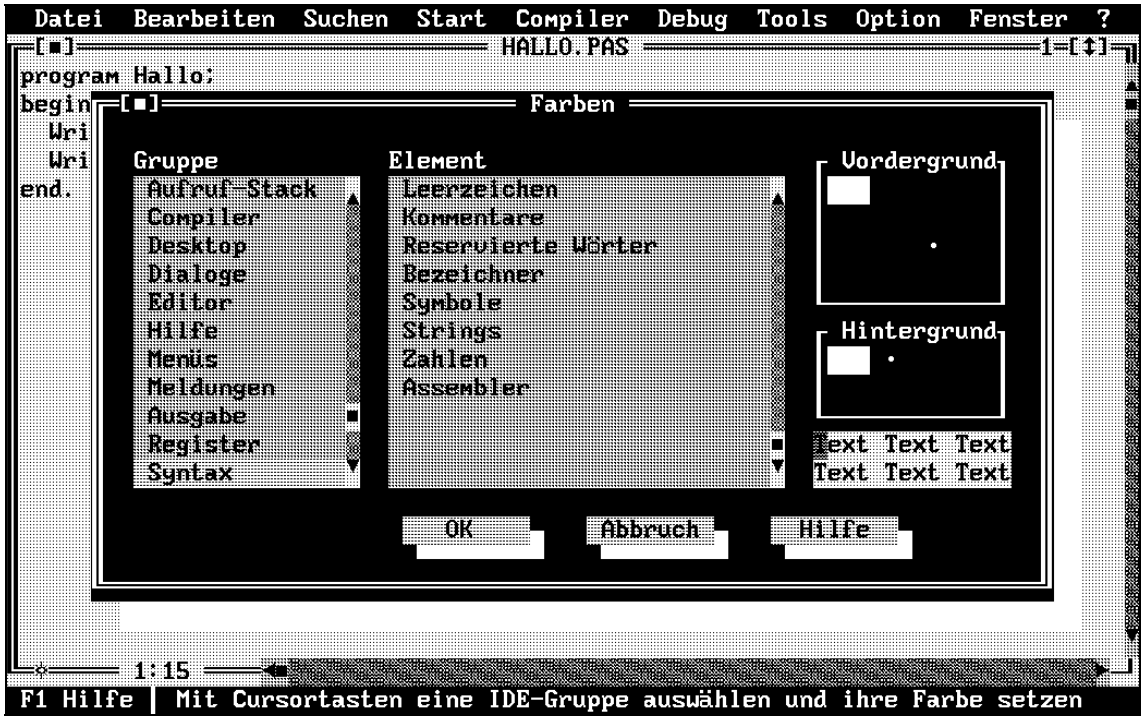

Abbildung 5-B26: Der Untermenüpunkt "Option/Umgebung/Farben..."

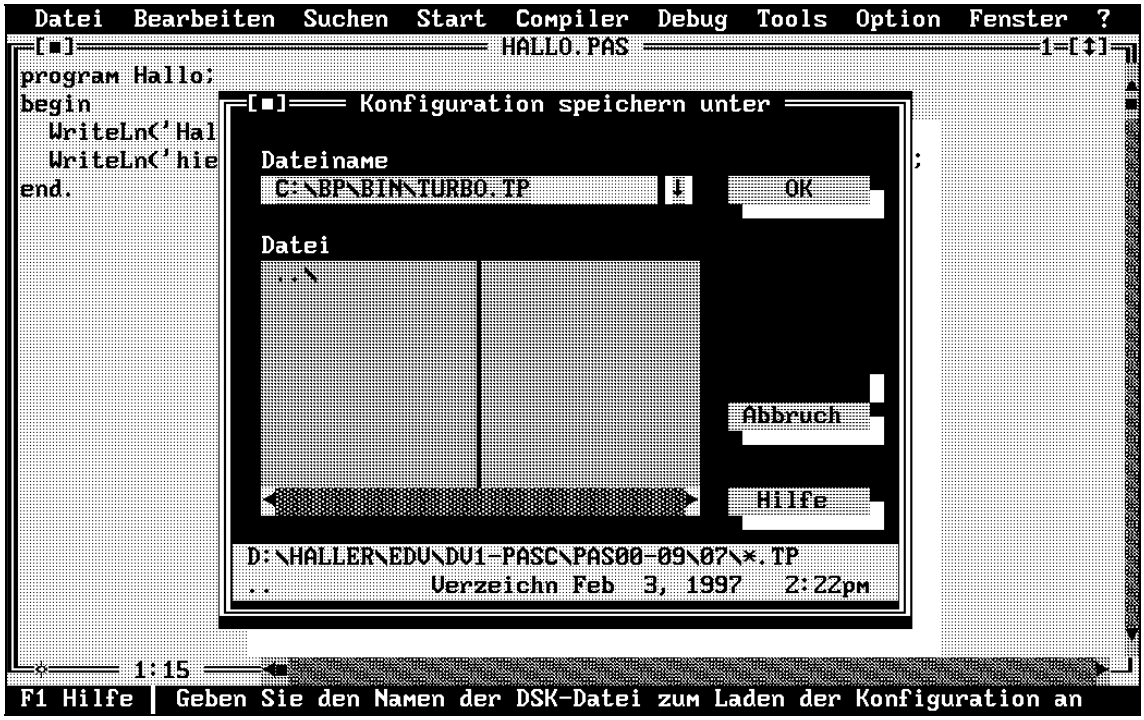

Abbildung 5-B27: Der Untermenüpunkt "Option/Speichern unter..."

5.10 Der Hauptmenüpunkt Fenster

Abbildung 5-B28: Der Haupmenüpunkt "Fenster"

In diesem Fenster kann man die Plazierung und die Größe der geöffneten Fenster ändern, zwischen Fenstern umschalten und ähnliches mehr. Der Autor ist der Meinung, daß im Normalfall die Fenstermanipulation mit der Maus schneller geht, wenn sich Fenster nicht gerade irgendwo verstecken. Der Untermenüpunkt "**Schließen**..." hat drei nachgesetzte Punkte. Das ist insofern falsch, als damit sonst in der IDE der Aufruf eines Dialogfensters signalisiert wird. Es erscheint aber keines, sondern es wird das aktuelle Fenster geschlossen, was man natürlich auch mit Schließfeld bewerkstelligen kann. Immerhin tut das Tastenkürzel "Alt+F3" kund, daß man damit auch das gleiche Ziel erreichen kann. Mit dem Untermenüpunkt "**Liste**.." erscheint das Dialogfenster "Fensterliste", in dem alle geöffneten Fenster aufgelistet sind. Durch Doppelklick auf den einen Fensternamen wird diesen zum aktuellen Fenster. Es können aber auch mit der Schaltfläche "Löschen" Fenster geschlossen werden. Bei Editfenstern erscheint eine Sicherheitsabfrage, wenn die Datei noch nicht gespeichert ist.

5.11 Der Hauptmenüpunkt Hilfe "?"

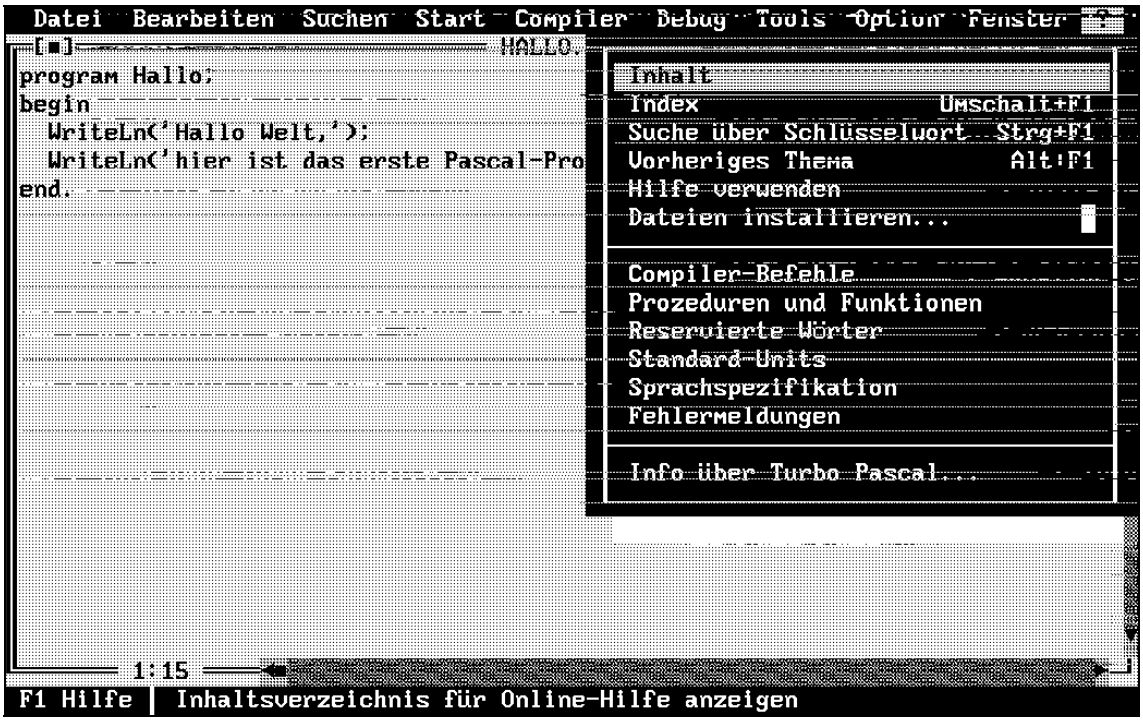

Abbildung 5-B29: Der Haupmenüpunkt Hilfe "?"

Dieser Hauptmenüpunkt enthält ein sehr gut ausgebautes Hilfe-System, in dem im Hypertextverfahren durch Doppelklick auf gelb hervorgehobene Begriffe sofort zu diesen verzweigt werden kann. Mit "Umsch+F1" kann ein Index aufgerufen werden, mit "Strg+F1" wird eine kontextbezogenen Hilfe zu dem Begriff angezeigt, in dem der Cursor steht. Beim Programmieren sehr praktisch, wenn man z.B. nicht mehr genau weis, ob die Standardfunktion "Sin" im Bogen- oder Gradmaß arbeitet. Man stellt den Cursor auf ein beliebiges Zeichen von "Sin" und drückt dann "Strg+F1". Mit "Alt+F1" kann der letztbenutzte Hilfebildschirm wieder angezeigt werden.

Für die Programmierpraxis ist der Untermenüpunkt "Fehlermeldungen" (sollte wie einige andere Untermenüpunkte auch drei nachgesetzte Punkte haben) sehr praktisch. Tip: Man vergrößere die Fenster mit dem Zoomfeld zum Vollbild, dann braucht man nicht umständlich scrollen. Man kann über alle Fehler (Compilerfehler und Laufzeitfehler) detaillierte Meldungen erhalten, wenn man die entsprechende Fehlernummer mit Doppelklick anwählt. Das Schnüffeln in den Listen ist aber auch ganz lehrreich!

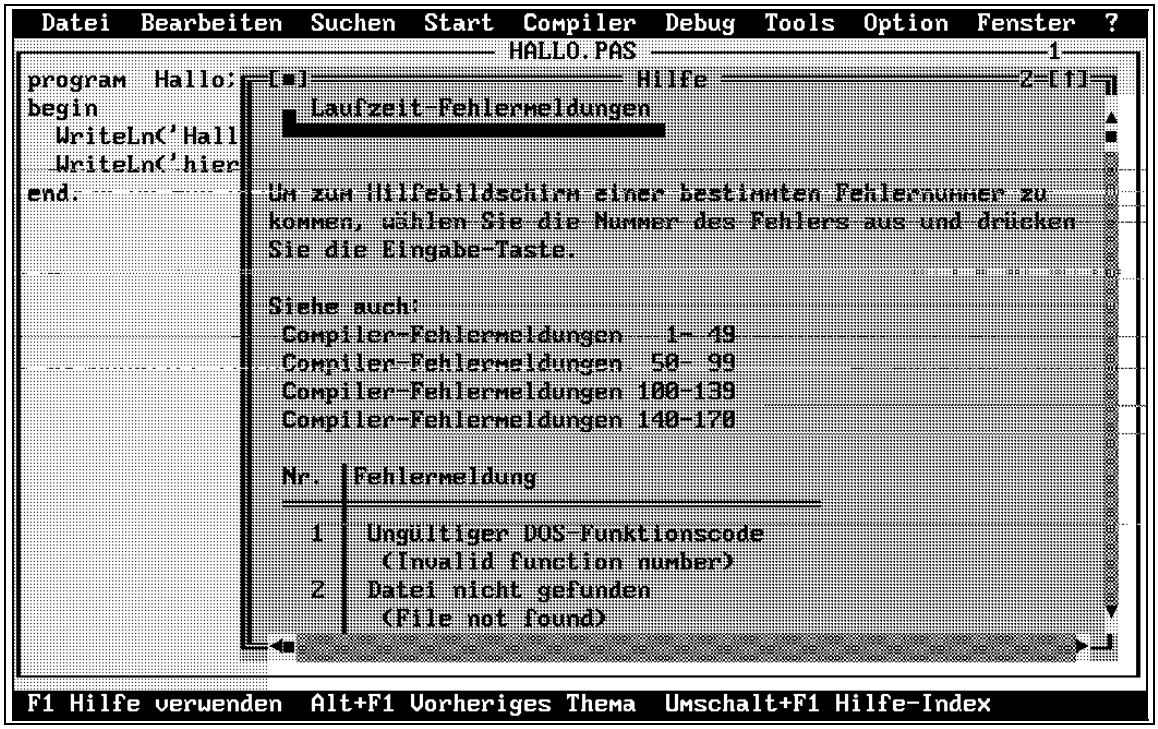

Abbildung 5-B30: Der Menüpunkt Hilfe "?/Fehlermeldungen/Laufzeit-Fehlermeldungen"
Durch Doppelklick auf "2" erhält man z.B. detaillierte Informationen zum Laufzeitfehler 2 (Datei nicht gefunden).

5.12 Die Compilerbefehle in Turbo-Pascal

Vorbemerkung: Dieser Unterpunkt wird nur wegen der Systematik bereits an dieser Stelle aufgeführt. Beim Erlernen der Sprache kann der Unterpunkt vorerst übersprungen werden. Bei späterem Bedarf ist hier nachzuarbeiten.

Compilerbefehle weisen den Compiler an, beim Übersetzen bestimmte Operationen auszuführen oder auch standardmäßig vorgesehene zu unterlassen. Zum Beispiel kann über einen Compilerbefehl die Stack-Speichergröße von einem Standardwert auf eine anderen Wert verändert werden, was z.B. bei rekursiven Aufrufen von Funktionen oder Prozeduren häufig notwendig ist.

Es gibt in der Systemumgebung IDE von Turbo-Pascal zwei Möglichkeiten, Compilerbefehle abzusetzen:

1. Als (Pseudo-) **Kommentare** in einer speziellen Schreibweise im Quelltext. Sie haben höhere Priorität als die eventuell nach Ziffer 2 gesetzte Compilerbefehle.
   Der Compilerbefehl beginnt mit einer öffnenden Kommentarklammer, dem Zeichen **{** oder dem Ersatzzeichen **(*** , es folgt ohne Leerzeichen das Dollarzeichen **$**, dann ein spezieller Buchstabe, der den eigentliche Befehlsnamen abkürzt und weitere Zeichen, je nach Befehl. Abgeschlossen wird der Compilerbefehl mit der

schließenden Kommentarklammer, dem Zeichen **}** oder dem Ersatzzeichen ***)**. Vor der schließenden Klammer dürfen noch "echte" Kommentare stehen; sie müssen aber von den Zeichen, die zum Compilerbefehl gehören, durch mindestens ein Leerzeichen abgetrennt sein.

Beispiel: Der Compilerbefehl, der die vollständige Auswertung von booleschen Ausdrücken erzwingt in zwei Schreibweisen; im zweiten Fall kombiniert mit einem zusätzlichen "echten" Kommentar".

```
{$B+}
{$B+  Vollständige Auswertung boolescher Ausdrücke }
```

2. Als Änderung der Standardeinstellung im Menü "Option/Compiler...", siehe Kap. 5.9. Von dieser Methode sollte man bei der Programmentwicklung im Normalfall keinen Gebrauch machen, da der Quelltext keinen Hinweis auf die gesetzten Compilerbefehl enthält.

Compilerbefehle werden genauer unterteilt in:

- Compilerschalter
- Compilerparameter
- Compilerbedingungen

Einfügen aller Compilerschalter und Compilerparameter in den Quelltext

Mit der Tastenkombination **Strg+O O** werden die Compilerschalter und Compilerparameter, die über den Menüpunkt "Option/Compiler..." eingestellt sind, vor den Kopf des Quelltextes geschrieben. Sie sind dann Bestandteil des Quelltextes und können auch editiert werden. Beim Verändern haben sie höhere Priorität als die Options-Einstellungen und werden mit dem Quelltext abgespeichert. Beispiel:

```
{$A+,B-,D+,E-,F-,G-,I+,L+,N+,O-,P+,Q+,R+,S+,T-,V+,X-}
{$M 16384,0,655360}
program Hallo;
begin
  WriteLn('Hallo Welt,');
  WriteLn('hier ist das erste Pascal-Programm von Anton Huber');
end.
```

5.12.1 **Compilerschalter** (in alphabetischer Auflistung)

Mit Comilerschaltern werden bestimmte Funktionen des Compilers ein- oder ausgeschaltet. Zum Einschalten dient das Pluszeichen **+** (Plus-Modus), zum Ausschalten das Minuszeichen **-** (Minus-Modus). Diese Schaltzeichen werden an den Befehlsbuchstaben angehängt.

Compilerschalter wirken entweder global, d.h. für die gesamte Compilierung oder lokal.

Globale Compilerschalter müssen grundsätzlich unmittelbar vor oder nach dem einleitenden **`program`** oder **`unit`** stehen, also auf jeden Fall vor dem Beginn der Deklarationen. Sie können nicht an anderer Stelle wieder aufgehoben werden.

Lokale Compilerschalter können an beliebiger Programmstelle stehen. Ein lokaler Compilerschalter bleibt solange wirksam, bis er durch den Schalter mit dem Minuszeichen wieder aufgehoben wird oder bis zum Ende des Programms.

Für jeden Schalter, gleich ob global oder lokal, gibt es eine Standardvorgabe, d.h. entweder Plus-Modus oder Minus-Modus.

Es können mehrere Compiler-Schalter in einer Kommentarklammer zusammengefaßt werden. Sie sind dann durch Kommas voneinander zu trennen. *Leerzeichen* und *echte Kommentare* sind zwischen den Schaltern *nicht erlaubt*, wohl aber am Ende. Die Reihenfolge der Schalter ist beliebig. Das Dollarzeichen wird in diesem Falle nur einmal angeschrieben. Beispiel: **`{$B+,S+,R-}`**

1. Compilerschalter A: Word-Datenausrichtung

Syntax: `{$A+}` oder `{$A-}`
Standardvorgabe: `{$A+}`
Typ: global
Menü-Äquivalent: Option/Compiler.../Word-Datenausrichtung

Mit diesem Schalter wird festgelegt, wie der Compiler Variablen im Speicher plaziert. Im Minus-Modus {$A-} findet keine spezielle Ausrichtung statt. Die Variablen werden so dicht wie möglich gepackt. Im Plus-Modus {$A+} werden die Variablen die mehr als 1 Byte belegen so im Speicher angeordnet, daß sie mit geradzahligen Speicheradressen beginnen. Der Speicherzugriff wird dadurch beschleunigt. Hat beim Prozessor i8088 (kennt heute keiner mehr) keine Wirkung. Hat bei Byte-Variablen, Array-Elemente und Record-Feldern keine Wirkung.

2. Compilerschalter B: Boolesche Ausdrücke vollständig

Syntax: `{$B+}` oder `{$B-}`
Standardvorgabe: `{$B-}`
Typ: lokal
Menü-Äquivalent: Option/Compiler.../Boolesche Ausdrücke vollständig

Mit diesem Schalter wird festgelegt, ob ein boolescher Ausdruck, der die logische Operatoren **`and`**, **`or`**, **`xor`** oder **`not`** enthält, vollständig ausgewertet werden soll oder nicht. Im Plus-Modus {$B+} werden *alle* Teile des booleschen Ausdrucks ausgewertet. Im Minus-Modus {$B-} wird die Auswertung dagegen abgebrochen, wenn das Endergebnis bereits feststeht. Zum Beispiel bei **`or`**, wenn ein Teilausdruck bereits `True` ergibt, oder bei **`and`**, wenn ein Teilausdruck bereits `False` ergibt. Der Minus-Modus (Kurzschlußmodus) erzeugt einen schnelleren Code. Turbo-Pascal berechnet boolesche Ausdrücke von links nach rechts. Etwas spitzfindig, aber es könnte sein, daß bei den restlichen Teilausdrücken selbstdefinierte Funktionen auftreten, in denen z.B. globale Variablen initialisiert werden; in diesem Fall ist der Plus-Modus unbedingt notwendig.

3. Compilerschalter D: Debug-Informationen

Syntax: `{$D+}` oder `{$D-}`
Standardvorgabe: `{$D+}`
Typ: global
Menü-Äquivalent: Option/Compiler.../Debug-Informationen

Im Plus-Modus {$D+} erzeugt der Compiler zusätzlichen Code zur Fehlersuche bei Laufzeitfehlern. Beim Compilieren zu einer EXE-Datei oder TPU-Datei wird der Zusatzcode in der EXE- bzw-TPU-Datei gespeichert. Nur in disem Modus können mit dem Menüpunkt "Suchen/Laufzeitfehler suchen..." Laufzeitfehler lokalisiert werden. Wenn im Arbeitspeicher compiliert wird, geschieht die Zuordnung von Laufzeitfehlern zum Quelltext im Plus-Modus automatisch. Dieser Schalter sollte mit dem Compilerschalter {$L+} kombiniert werden.

4. Compilerschalter E: Emulation Coprozessor 80x87

Syntax: `{$E+}` oder `{$E-}`
Standardvorgabe: `{$E+}`
Typ: global
Menü-Äquivalent: Option/Compiler.../Emulation

Der Compilerschalter E ist ggf. mit dem Compilerschalter N (siehe dort) zu kombinieren. Sinnvolle Kombinationen sind:

`{$N+,E-}` Programm setzt unbedingt Coprozessor voraus

`{$N+,E+}` Programm ist auf allen Systemen lauffähig, egal ob Coprozessor existiert ist oder nicht. Wenn der Coprozessor existiert, wird dieser benutzt und nicht die (wesentlich langsamere) Emulation.

Im Plus-Modus {$E+} können auch die zusätzlichen Real-Datentypen genutzt werden, die sonst einen Coprozessor voraussetzen, das sind:

- Single 4 Byte
- Double 8 Byte
- Extended 10 Byte
- Comp 8 Byte

Siehe auch späteres Programm "Pas08031.PAS" im Kap. 8.

5. Compilerschalter F: Far-Aufrufe erzeugen

Syntax: `{$F+}` oder `{$F-}`
Standardvorgabe: `{$F-}`
Typ: lokal
Menü-Äquivalent: Option/Compiler.../Far-Aufrufe erzeugen

Die Mikroprozessoren der MS-DOS-PCs arbeiten mit segmentierter Speicheradressierung. Eine Speicheradresse setzt sich aus den beiden Teilen Segment-Adresse und Offset-Adresse zusammen. Ein Segment umfaßt max. 64 KByte. Man unterscheidet auf Maschinensprach-Ebene zwischen **Near**-Aufrufen (nah) und **Far**-Aufrufen (fern) von Routinen. Bei **Near** laufen Aufrufe und Rücksprünge innerhalb eines einzigen Code-Segments ab; es muß deshalb nur die Offset-Adresse auf den Stack gelegt werden. Bei **Far** kann dagegen mit verschiedenen Segmenten gearbeitet werden. Es muß dann aber auch die Segment-Adresse auf den Stackspeicher gelegt werden. Bei {$F+} werden die Adressen aller Routinen (Prozeduren und Funktionen) als **Far** mit Segment:Offset-Zeigern übergeben; bei der nichtaktivierten Standardeinstellung **Near** dagegen nur mit der Offsetadresse. **Far**-Aufrufe sind i.a. nur bei der Übergabe von

Routinen als Parameter an andere Routinen zwingend, siehe Kap. 11.9. Turbo-Pascal hat Standardeinstellungen wie folgt:

- Wenn die Routine im Hauptprogramm oder im Implementationsteil einer Unit deklariert ist, wird sie als **Near** codiert. Siehe Kap. 23: Units in Turbo-Pascal
- Wenn die Routine im Interface-Teil einer Unit deklariert ist, wird sie als **Far** codiert.

6. Compilerschalter G: 286-Instruktionen erzeugen

Syntax: `{$G+}` oder `{$G-}`
Standardvorgabe: `{$G-}` für Real-Mode, sonst `{$G+}`
Typ: global
Menü-Äquivalent: Option/Compiler.../286-Instruktionen erzeugen

Im Plus-Modis {$G+} laufen die Programme auf Systemen mit Prozessor ab dem i80286 etwas schneller, bei den Uralt-Prozessoren i8088 und i8086 dagegen nicht mehr.

7. Compilerschalter I: I/O-Prüfung

Syntax: `{$I+}` oder `{$I-}`
Standardvorgabe: `{$I+}`
Typ: lokal
Menü-Äquivalent: Option/Compiler.../IO-Prüfung

Im Plus-Modus {$I+} werden Ein- und Ausgaben (Tastastur, Bildschirm, Datei, Drucker usw.) auf Fehler geprüft. Im Fehlerfall wird mit einem Laufzeitfehler abgebrochen; der Anwender steht im Freien. Ein Trivialbespiel: Einlesen einer numerischen Variablen von der Tastatur und die Eingabe enthält ein nichtnumerisches Zeichen. Wenn man eine eigene Fehlerbehandlung programmieren möchte, um z.B. einen Absturz des Programms durch Bedienungsfehler zu vermeiden, ist vor der Ein- oder Ausgabe der Minus-Modus {$I-} zu setzen. Nach der Ein-/Ausgabeoperation ist unbedingt der Compilerbefehl wieder auf den Plus-Modus {$I+} zu setzen. Die Fehlerbehandlung ist mit der Standardfunktion `IOResult` (liefert bei Fehlerfreitheit den Wert 0) zu programmieren, sinnvollerweise in einer repeat/until-Schleife, siehe Kap. 24: Fehlerbehandlung und Fehlersuche.

8. Compilerschalter K: Smart Callbacks (Protected-Mode, Ausgabeziel Windows)

Syntax: `{$K+}` oder `{$K-}`
Standardvorgabe: `{$K+}`
Typ: global
Menü-Äquivalent: Option/Compiler.../Smart Callbacks

Der Schalter K steht nur im Protected-Mode (Borland-Pascal BP.EXE, nicht Turbo-Pascal Turbo.EXE) zur Verfügung, wenn als Ausgabeziel (Zielplattform) Windows gewählt wurde. Für Turbo-Pascal ohne Belang.

9. Compilerschalter L: Lokale Symbole

Syntax: `{$L+}` oder `{$L-}`
Standardvorgabe: `{$L+}`
Typ: global
Menü-Äquivalent: Option/Compiler.../Lokale Symbole

Der Schalter legt fest, ob neben den bei {$D+} (, 5.12.1.3) erzeugten Debug-Informationen zusätzliche Informationen über *lokale Symbole* in ein Modul aufgenommen werden {$L+} oder nicht {$L-}. Wichtig für den Einsatz des internen oder externen Turbo-Debuggers. Nur wirksam mit dem Compilerschalter {$D+}. Erst mit beiden aktivierten Checkboxen zeigt der Aufruf des Menüs "Debug/Aufruf-Stack" Wirkung.

10. Compilerschalter N: Numerischer Coprozessor

Syntax: {$N+} oder {$N-}
Standardvorgabe: {$N-}
Typ: global
Menü-Äquivalent: Option/Compiler.../80x87-Code

Im Minus-Modus {$N-} steht von den Real-Typen nur der Standard-Realtyp (6 Byte) zur Verfügung. Der Plus-Modus {$N+} ist nur möglich, wenn der Rechner mit einem mathematischen Coprozessor (z.B. Intel 8087, 80287, 80387 oder Prozessoren mit integrierten Coprozessor wie 80486 DX oder Pentium) ausgestattet ist. Gleitkomma-Operationen laufen mit Coprozessor ungleich schneller ab. Es stehen dann auch die zusätzliche Real-Datentypen zur Verfügung:

- Single 4 Byte
- Double 8 Byte
- Extended 10 Byte
- Comp 8 Byte

Anmerkung: Der Coprozessor kann mit dem Compilerschalter {$E+} (siehe dort) auch softwaremäßig emuliert werden; die Emulation ist aber erheblich langsamer. Aber auch bei der Emulation können die zusätzlichen Real-Datentypen genutzt werden. Die Kombination {$N+,E+} ist auf allen Systemen lauffähig, egal ob ein Coprozessor existiert oder nicht. Wenn der Coprozessor existiert, wird dieser benutzt und nicht die Emulation. Die Kombination {$N+,E-} setzt den Coprozessor zwingend voraus, erzeugt aber einen kleineren EXE-File. Siehe Programm "Pas08031.PAS" im Kap. 8.

11. Compilerschalter O: Overlay-Prüfung (nur Real-Mode)

Syntax: {$O+} oder {$O-}
Standardvorgabe: {$O-}
Typ: global
Menü-Äquivalent: Options/Compiler.../Overlays möglich

Im Plus-Modus {$O+} können Units bei Bedarf Overlay-fähig gemacht werden (sie können aber auch als "normale" Units verwendet werden), im Minus-Modus nicht. Zu Overlays siehe Kap. 23: Include-Dateien, Overlays und Units.

12. Compilerschalter P: Offene Array-Grenzen

Syntax: {$P+} oder {$P-}
Standardvorgabe: {$P-}
Typ: lokal
Menü-Äquivalent: Option/Compiler.../Offene Array-Grenzen

Im Plus-Modus {$P+} ermöglicht die **Deklaration von offenen Arrays oder offenen Strings** (mit neuem Datentyp "OpenString") als formale Parameter in Routinen (Funktionen und Prozeduren). Damit können an die gleichen Routinen Arrays oder Strings mit verschiedenen Größen übergeben werden. Im aufrufenden Programmteil müssen aber die Arrays und Strings in konventioneller Weise deklariert sein. Die Übergabe kann wie sonst auch mit Wert oder Adresse erfolgen. Die Indizies der übergebenen Arrays

oder Strings laufen aber innerhalb der Routine immer von 0 bis n-1. Der kleinste und der größte Index können mit den Standardfunktionen `Low(`*`arraybezeichner`*`)`und `High(`*`arraybezeichner`*`)` ermittelt werden. Innerhalb der Routine liefert `Low` immer den nicht sehr interessanten Wert 0, `High` dagegen die Anzahl der Elemente minus 1. Außerhalb der Routine liefern beide Funktionen dagegen die deklarierten Array- bzw. Stringgrenzen. Offene Arrays können aber im Gegensatz zu normalen Arrays nur elementweise verarbeitet werden, nicht als Ganzes. Beispiel unter Verwendung des neuen Datentyps "OpenString".

```
program Test;
var
  x: array[2..7]   of Byte;
  y: array[-2..47] of Byte;
  s: string[5];
  t: string[80];
...
procedure DemoOffeneArrays(var x: array of Byte; s: OpenString);
begin                     { x mit Adress-, s mit Werte-Übergabe }
  ...
  ...
end;

begin { Hauptprogramm }
  ...
  { Hier müßte noch die Initialisierung stehen }
  ...
  DemoOffeneArrays(x, s);
  DemoOffeneArrays(y, t);
  DemoOffeneArrays(x, 'Huber Toni');
  ...
end.
```

13. Compilerschalter Q: Überlaufprüfung

Syntax: `{$Q+}` oder `{$Q-}`
Standardvorgabe: `{$Q-}`
Typ: lokal
Menü-Äquivalent: Option/Compiler.../Überlaufprüfung

Im Plus-Modus {$Q+} wird bei Operationen mit Ordinal- und Unterbereichstypen geprüft, ob das Ergebnis im zulässigen Bereich liegt, z.B. beim Datentyp Integer zwischen den Grenzen-32768 und +32767. Bei den Operationen können auch die Standardfunktionen "`Abs Sqr Succ Pred`" vorkommen. Im Fehlerfall wird mit einem Laufzeitfehler abgebrochen. Leider werden jedoch die Standardfunktionen"`Inc`" und "`Dec`" nicht geprüft. Diese Option ist dennoch eine notwendige Ergänzung zum Plus-Modus des Compilerbefehls {$R+} "Bereichsprüfung".

14. Compilerschalter R: Bereichsüberprüfung

Syntax: `{$R+}` oder `{$R-}`
Standardvorgabe: `{$R-}`
Typ: lokal.
Menü-Äquivalent: Option/Compiler.../Bereichsüberprüfung

Im Plus-Modus {$R+} (R steht für Range) werden die Indizies von Arrays und Strings auf zulässige Werte geprüft, ebenso ob bei ***Zuweisungen*** an Ordinaltypen und Unterbereichstypen die Werte in den zulässigen Grenzen liegen, bei Integern z.B. zwischen -32768 und +32767. Im Fehlerfall bricht das Programm mit einem Laufzeitfehler ab. Mit dieser Option allein wird aber nicht geprüft, ob bei ***Operationen*** mit diesen Typen Fehler vorliegen; dafür ist der Compilerschalter {$Q+} "Überlaufprüfung" notwendig. Der Plus-Modus {$R+} vergrößert den Code und die Ausführungszeit. Dennoch sollte man den Plus-Modus setzen, da sonst im Fehlerfall keine Reaktion erfolgt. Das Programm rechnet im Minus-Modus im Fehlerfall ohne Warnung mit falschen Daten weiter.

15. Compilerschalter S: Stack-Prüfung

Syntax: `{$S+}` oder `{$S-}`
Standardvorgabe: `{$S+}`
Typ: lokal
Menü-Äquivalent: Option/Compiler.../Stack-Prüfung

Im Plus-Modus {$S+} wird vor jedem Aufruf einer Funktion oder Prozedur überprüft, ob auf dem Stack-Speicher noch genügend Platz zur Speicherung der lokalen Variablen und der Rücksprung-Adresse vorhanden ist. Wenn nicht, wird das Programm mit einer entsprechenden Fehlermeldung abgebrochen. Mit dem später behandelten Compilerparameter `{$M ...,...,...}` kann man die Größe des Stackspeichers verändern. Der Plus-Modus sollte zumindest in der Testphase eingeschaltet sein (oder bleiben), da Stack-Überläufe oft zu System-Abstürzen führen. Der Plus-Modus erhöht aber den Umfang des Programms und setzt auch die Ausführungsgeschwindigkeit herab. Beim Minus-Modus {$S-} kann es im Fehlerfall zum Systemabsturz kommen.

16. Compilerschalter T: Typisierter @-Operator

Syntax: `{$T+}` oder `{$T-}`
Standardvorgabe: `{$T+}`
Typ: global
Menü-Äquivalent: Option/Compiler.../Typisierter @-Operator

Der Adressoperator `@` liefert allgemein die Adresse einer Variablen, einer Prozedur oder einer Funktion als Zeiger, z.B. liefert `@x` oder `@(x)` den Zeiger auf die angenommene Variable x. Im Plus-Modus {$T+} sind bei Referenz auf eine Variable die zurückgelieferten Zeiger immer Zeiger vom Typ der Variablen; bei Referenz auf Prozeduren oder Funktionen aber immer ein Pointer-Typ, d.h. ein untypisierter Zeiger. Im Minus-Modus {$T-} ist der zurückgelieferte Zeiger dagegen immer ein untypisierter Zeiger.

17. Compilerschalter V: Überprüfung von var-Strings

Syntax: `{$V+}` oder `{$V-}`
Standardvorgabe: `{$V+}`
Typ: lokal
Menü-Äquivalent: Option/Compiler.../Strenge Prüfung von VAR-Strings

Im Plus-Modus {$V+} überprüft der Compiler bei der Übergabe von var-Strings an Prozeduren oder Funktionen (var-Übergabe mit Adresse statt mit Wert), ob die formalen und aktuellen Strings die gleichen Längenvereinbarung besitzen. Im Fehlerfall wird mit einer Fehlermeldung abgebrochen. Im Minus-Modus {$V-} kann jede Stringvariable als var-Parameter übergeben werden, auch wenn die Längen des formalen und des aktuellen Strings verschieden deklariert sind, was zwar praktisch ist, dennoch sollte der

Programmierer ein ganz schlechtes Gewissen haben! Besser: den Datentyp OpenString verwenden (ab Turbo-Pascal 7.0).

18. Compilerschalter W: Windows Stack Frames (Protected-Mode, Ausgabeziel Windows)

Syntax: `{$W+}` oder `{$W-}`
Standardvorgabe: `{$W+}`
Typ: global
Menü-Äquivalent: Option/Compiler.../Windows-Stack-Frame-erzeugen

Der Schalter W steht nur im Protected-Mode (Borland-Pascal BP.EXE, nicht Turbo-Pascal Turbo.EXE) zur Verfügung, wenn als Ausgabeziel (Zielplattform) Windows gewählt wurde. Für Turbo-Pascal nicht von Belang.

19. Compilerschalter X: Erweiterte Syntax

Syntax: `{$X+}` oder `{$X-}`
Standardvorgabe: `{$X-}`
Typ: global
Menü-Äquivalent: Option/Compiler.../Erweiterte Syntax

Für die Grundausbildung ist der Minus-Modus {$X-} einzustellen. Nicht weitersagen: Ab Turbo Pascal 6.0 können Funktionen *zusätzlich* auch im Sinne von Prozeduren verwendet werden, siehe Kap. 11. Die Funktionsrückgabewerte werden dann im gegebenen Fall einfach weggelassen.. Das ist zwar fast C-like, aber absolut nicht im Sinne von Nikolaus Wirth und für den Anfänger eine ziemliche Verwirrung. Der Autor läßt diese Option für die Pascal-Grundausbildung nicht zu, auch wenn er sie heimlich ab und zu mal selbst benutzt! Dennoch: Die ab Borland Pascal 7.0 via **Unit "Strings**" möglichen **nullterminierten Strings** (max. 65535 Zeichen, Längenbegrenzung mit dem Null-Byte, gegenüber 255 Zeichen beim Standard, Längenbyte als Zeichen Nr. 0) sind **nur mit "Erweiterter Syntax"** möglich.

20. Compilerschalter Y: Symbolreferenz (Protected Mode)

Syntax: `{$Y+}` oder `{$Y-}`
Standardvorgabe: `{$Y-}`
Typ: global
Menü-Äquivalent: Option/Compiler.../Symbol-Information

Der Schalter W steht nur im Protected-Mode (Borland-Pascal BP.EXE, nicht Turbo-Pascal Turbo.EXE) zur Verfügung. Der Plus-Modus {$Y+} generiert Symbol-Referenzinformationen in einem compilierten Programm oder in einer Unit, so daß die Symbolanzeige (bei Borland Pascal für den Protected-Mode über den Menüpunkt "Option/Symbole...") Symboldefinitionen und Referenzinformationen anzeigen kann.

5.12.2 Compilerparameter

Mit Parametern werden Zahlenwerte wie z.B. Speichergrößen oder Texte wie z.B. Dateinamen festgelegt. Für die zahlenmäßigen Compilerparameter stehen Standardwerte zur Verfügung. Im Gegensatz zu Compilerschaltern **muß** bei Compilerparametern nach

dem Befehlsbuchstaben ein **Leerzeichen** folgen, nach den Parametern können noch "echte" Kommentare folgen. Beispiele:

```
{$M 65000,8000,650000  Memory-Aufteilung ändern  }
{$L Test.OBJ  Einbinden einer Objekt-Datei mit dem Namem
              "Test.OBJ" beim Linken des Programms }
```

Compilerparameter wirken teils global, teils lokal, siehe folgende Auflistung. Globale Compilerparameter müssen wie Compilerschalter vor dem einleitenden **`program`** oder nach dem einleitenden **`program`** oder **`unit`** bzw. vor **`uses`**, **`label`**, **`const`**, **`type`**, **`var`**, **`function`**, **`procedure`** und **`begin`** stehen.

1. Compilerparameter D: Programmbescheibung (nicht beim Real-Mode)

Syntax: `{$D text}`
Standardvorgabe: keine
Typ: global
Menü-Äquivalent: kein

Mit diesem Parameter wird der Compiler angewiesen, den angebenen Text in die EXE-Datei oder DLL-Datei mit aufzunehmen. Nicht für Units verwenden. Beispiel:

```
{$D Das Superprogramm von Anton Huber, Version 47.11}
```

2. Compilerparameter G: Unit-Segmente gruppieren (nicht für Real-Mode)

Syntax: `{$G unit1, unit1, ... }`
Standardvorgabe: keine
Typ: global
Menü-Äquivalent: kein

Mit diesem Parameter wird der Compiler angewiesen, die angebenen Units in ein einziges Segment zu linken.

3. Compilerparameter I: Include-Datei einlesen

Syntax: `{$I dateiname }`
Standardvorgabe: keine
Typ: lokal
Menü-Äquivalent: kein

Mit diesem Parameter wird der Compiler angewiesen, die angegebene Pascal-Quelltext-Datei an der Stelle des Compilerparameters einzulesen und zu compilieren. Die einzulesende Datei muß abgeschlossene Programmblöcke enthalten (Prozeduren und Funktionen) und/oder ein vollständiges Hauptprogramm mit **`begin`** und **`end`**. Wenn *`dateiname`* keine Extension aufweist, wird standardmäßig `.PAS` angenommen. Wenn *`dateiname`* keinen Zugriffspfad enthält, dann wird zuerst das momentan gesetzte Verzeichnis durchsucht, anschließend alle Verzeichnisse, die mit der Menü-Auswahl *Option/Verzeichnisse/Include-Verzeichnisse* gesetzt sind. Include-Dateien können bis zu einer Tiefe von 15 Ebenen verschachtelt sein; d.h. eine Include-Datei kann auch eine

weitere Include-Datei aufrufen. Details siehe Kap. 23: Include-Dateien, Overlays und Units. Beispiel: `{$I C:\Student\LiGleich.PAS}`

4. Compilerparameter L: Object-Dateien einbinden

Syntax: `{$L dateiname }`
Standardvorgabe: keine
Typ: lokal
Menü-Äquivalent: kein

Mit diesem Parameter wird der Linker angewiesen, die angegebene Object-Datei einzubinden. Wenn *dateiname* keine Extension aufweist, wird standardmäßig `.OBJ` angenommen. Wenn *dateiname* keinen Zugriffspfad enthält, dann wird zuerst das aktuelle Verzeichnis durchsucht, anschließend alle Directories, die mit der Menü-Auswahl *Option/Verzeichnisse.../Objekt-Verzeichnisse* gesetzt sind.

5. Compilerparameter M: Speicherbelegung ändern

Syntax: `{$M Stackgröße,HeapMin,HeapMax }` Real-Mode
Syntax: `{$M Stackgröße }` Protected-Mode
Standardvorgabe: `{$M 16384,0,655360 }` Wert 655360 für DOS-Speicher 640 KB
Typ: global
Menü-Äquivalent: Option/Compiler../Speicherauslegung

Stackgröße: 1024..65520, Standard 16384
HeapMin, HeapMax: 0..655360 (bei Speicher 640 KByte)
HeapMin <= HeapMax
Bei Units hat dieser Compilerparameter keine Wirkung.

6. Compilerparameter O: Overlay-Deklaration

Parametername: Overlay
Syntax: `{$O unitname }`
Standardvorgabe: keine
Typ: lokal
Menü-Äquivalent: kein

Dieser Compiler-Parameter deklariert eine Unit als Overlay. Damit wird der Code nicht direkt in das Programm eingebunden, sondern in einer Datei mit der Extension `.OVR` gespeichert. Das mit *unitname* gemeinte Unit muß zuvor mit **`uses`** aufgeführt sein und mit dem Plus-Modus des Compilerschalters {$O+} compiliert worden sein.

5.12.3 Compilerbedingungen

Mit einer Compilerbedingung kann festgelegt werden, ob bestimmte Teile des Quelltextes in Abhängigkeit von einer Bedingung compiliert werden sollen. In eingeschränk-

ter Form können Compilerbedingungen auch in das Eingabefeld "Definition für bedingte Compilierung" des Menüs "Option/Compiler..." geschrieben werden. Davon ist abzuraten.

Compilerbedingungen haben folgende Grundformen:

```
{$If bbbb tttt}
  ...
..{ "normale" Pascal-Teile }
  ....
{$ENDIF}
```

Für {$If *bbbb tttt*} kann stehen:

{$IFDEF symbolname}	Wenn *symbolname* definiert
{$IFNDEF symbolname}	Wenn *symbolname* nicht definiert
{$IFOPT compilerschalter}	Wenn *compilerschalter* gesetzt Beispiel: {$IFOPT N+}

Der zwischen {$If*bbbb ttt*} und {$ENDIF} liegende Programmteil wird nur dann compiliert, wenn die Bedingung *bbbb* zutrifft.

Die zweite Form ist mit einem else-Zweig eine Erweiterung der ersten Form:

```
{$If bbbb ssss}
....
....
{$ELSE}
....
....
{$ENDIF}
```

Darüber hinaus sind zwei weitere Compilerbedingungen definiert:

{**$DEFINE** *symbolname*}	Definiere eigenes Symbol. Die Symbole können frei gewählt werden so wie andere Bezeichner auch. Sie kollodieren aber nicht mit gleichlautenden Bezeichnern des eigentlichen Pascal-Programms.
{**$UNDEF** *symbolname*}	Lösche (undefine) eigenes Symbol

Turbo-Pascal besitzt folgende **vordefinierte Symbole**:

VER70	Für Borland Pascal Version 7.0 definiert; bei anderen Versionen entsprechend, z.B. VER50 für Version 5.0.
MSDOS	Für Betriebssystem MS-DOS im Real-Mode definiert.
WINDOWS	Für Betriebssystem MS-Windows definiert.
CPU86	Immer definiert. Für alle Intel-Prozessoren der Baureihe i80x86 und auch neuere wie Pentium und Pentium II.
CPU87	Ist definiert, wenn der Rechner über einen Coprozessor verfügt.

DPMI Für Betriebssystem MS-DOS im Protected-Mode definiert. DPMI = DOS Protected Mode Interface.

1. Beispiel:

```
....
{$DEFINE Testen}
....
{$IFDEF Testen}
  WriteLn('Das Symbol "Testen" ist definiert);
{$ENDIF}
....
```

2. Beispiel: Anwendung des Symbols **CPU87** bei bedingter Compilierung:

```
....
{$IFDEF CPU87}
  {$N+}             { Wenn der Coprozessor existiert, werden alle }
  type              { Realtypen durch den Typ Double ersetzt und  }
    Real = Double;  { somit mit doppelter Genauigkeit berechnet   }
{$ELSE}
  {$N-}                 { Wenn nicht, dann werden die Zusatz-     }
  type                  { Realtypen des Coprozesssors durch den   }
    Single   = Real;    { Standard-Realtyp ersetzt. Es wird dann  }
    Double   = Real;    { mit einfacher Genauigkeit gerechnet.    }
    Extended = Real;
    Comp     = Real;
{$ENDIF}
....
```