PC·NEWS

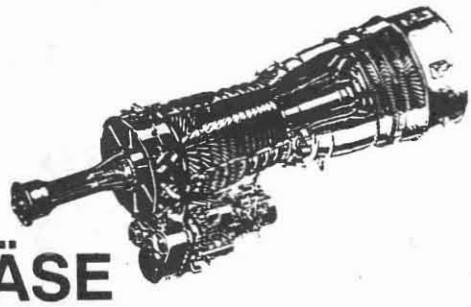
Das offizielle Mitteilungsblatt des

PCC-TGM

(Personal Computer Club - Technologisches Gewerbe-Museum)

. . . . .

TURBINE



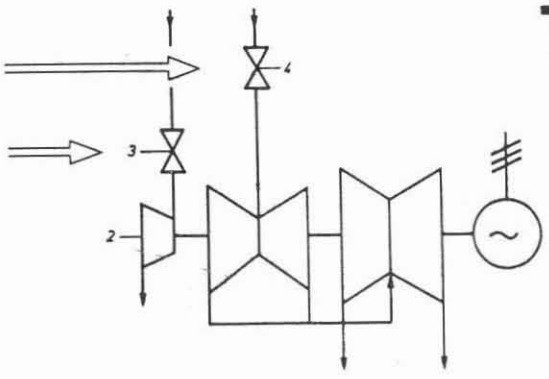
TURBO DIESEL

TURBO GEBLÄSE

TURBO Huber

TURBO KOMPRESSOR

TURBO LADER



TURBO PASCAL

TURBPOPROP

TURBULENZ

. . . . . .

7) Prozedur- und Funktionsdeklarationen / Beispiele:

```
PROCEDURE ADDW (Op1: WORD; VAR Op2: WORD); | Aufruf:              | FUNCTION ADDW (Op1,Op2: WORD): WORD; | Aufruf:
<lok. TYPE/CONST/VAR/PROC./FUNC.-Def.>     |   Op1:=5; Op2:=4;    | <lok. TYPE/CONST/VAR/PROC./FUNC.-Def>|   Op1:=5; Op2:=4; Op3:=0;
BEGIN                                      |   ADDW (Op1,Op2);    | BEGIN                                |   Op3:=ADDW (Op1,Op2);
  Op2:=Op1+Op2;                            |   WriteLn (Op1); -> 5 |   ADDW:=Op1+Op2;                     |   WriteLn (Op1); -> 5
END; (* PROCEDURE ADDW *)                   |   WriteLn (Op2); -> 9 | END; (* FUNCTION ADDW *)             |   WriteLn (Op2); -> 4
                                           |                       |                                      |   WriteLn (Op3); -> 9
```

8) Programme und Units:

```
Bsp Prg:PROGRAM Test;
          USES  <Liste der benutzten Units>
          TYPE  <globale Typ-Definit.>
          VAR   <globale Variablendefinit.>
          CONST <globale Konstantendefinit.>
          <Includes>
          <Prozedur- und Funktionsdef.>
        BEGIN
          <Source>
        END. (* Test *)
```

```
Bsp Unit:   UNIT Example;
            INTERFACE
              USES <Liste der benutzten Units>
              <öffentliche Deklarationen> z.B. PROCEDURE Bsp (VAR Lsg: STRING);
            IMPLEMENTATION
              <nicht öffentliche Deklarationen> z.B. PROCEDURE Bsp;
                                                       <Source>
                                                     END; (* PROC Bsp *)
            BEGIN
              <Source - z.B. Initialisierung aller öffentl. Var.>
            END. (* UNIT Example *)
```

9) Standardprozeduren und Funktionen:

D I E P R O Z E D U R E N E X I T U N D H A L T:

```
Exit        beendet die Ausf. des mom. Blocks mit sofortiger Wirkung;            Dekl: Exit;
            (Block: PROCEDURE  -> Sprung in die aufrufende Routine / Hauptprogr -> Programm wird beendet (wie HALT))
Halt        beendet Prgr. u. springt in den übergeordneten Prozess;              Dekl: Halt;
```

D Y N A M I S C H E V E R W A L T U N G D E S S P E I C H E R S:

```
Dispose     gibt den Speicherplatz einer dyn. Var. frei;                         Dekl: Dispose (VAR p: Pointer);
FreeMem     gibt einen dyn. bel. Speicher best. Gr. frei;                        Dekl: FreeMem (VAR p: Pointer; size: WORD);
GetMem      bel. Speicher best. Gr. f. dyn. Var., setzt Zeiger auf d. Anf. d. Ber.; Dekl: GetMem (VAR p: Pointer; size: WORD);
Mark        hält den Zust. des Heaps in einer Zeigervariablen fest;              Dekl: Mark (VAR p: Pointer);
MaxAvail    liefert (lft.) d. Gr. d. grten freien Heap-Bereichs, ermittelt,      Dekl: MaxAvail;            Erg: LONGINT;
            wieviele BYTEs momentan max. für eine dyn. Var. belegt werden können;
MemAvail    lft. die Gesamtzahl der freien BYTEs auf dem Heap;                   Dekl: MemAvail;            Erg: LONGINT;
New         erz. dyn. Var., setzt Zeiger auf d. Anf. des durch diese Var. bel. Ber.;Dekl: New (VAR p: Pointer);
Release     setzt den Heap auf den mit MARK festgehaltenen Zust. zurück;         Dekl: Release (VAR p: Pointer);
```

T R A N S F E R F U N K T I O N E N:

Chr	lft. Zeichen mit entspr. Ordinalzahl zurück;	Dekl: Chr (x: BYTE);	Erg: Char;
Ord	lft. die Ordinalzahl des angeg. (ordinalen) Wertes;	Dekl: Ord (x);	Erg: LONGINT;
Round	konvertiert REALwert in einen LONGINT, rundet auf oder ab;	Dekl: Round (x: REAL);	Erg: LONGINT;
Trunc	konv. REAL in einen LONGINT, Nachkommast. werden abgeschnitten;	Dekl: Trunc (x: REAL);	Erg: LONGINT;

A R I T H M E T I S C H E F U N K T I O N E N:

Abs	lft. den absoluten Wert des Arguments;	Dekl: Abs (x);	Erg: derselbe Typ wie Arg.;
Arctan	lft. den Arcustangens des Arg.;	Dekl: ArcTan (x: REAL);	Erg: REAL;
Cos	lft. den Cosinus des Arg.;	Dekl: Cos (x: REAL);	Erg: REAL;
Exp	e=2.7182..., liefert e hoch Arg.;	Dekl: Exp (x: REAL);	Erg: REAL;
Frac	lft. die Nachkommastellen;	Dekl: Frac (x: REAL);	Erg: REAL;
Int	lft. ganzzahligen Anteil des Arg.;	Dekl: Int (x: REAL);	Erg: REAL;
Ln	lft. den natürlichen Logarithmus des Arg.;	Dekl: Ln (x: REAL);	Erg: REAL;
Pi	lft. den Wert Pi (3.141592...) zurück;	Dekl: Pi;	Erg: 3.14..... (REAL);
Sin	lft. den Wert des Arg. zurück;	Dekl: Sin (x: REAL);	Erg: REAL;
Sqr	lft. das Quadrat des Arg. zurück;	Dekl: Sqr (x);	Erg: hat selben Typ wie Arg.;

O R D I N A L E P R O Z E D U R E N U N D F U N K T I O N E N:

Dec	erniedrigt Variable;	Dekl: Dec (x [,n]);	Erg: x:=x-1 oder x:=x-n;
Inc	erhöht Variable;	Dekl: Inc (x [,n]);	Erg: x:=x+1 oder x:=x+n;
Odd	prüft ob das Arg. ein unger. Zahl ist (ung->TRUE);	Dekl: Odd (x: LONGINT);	Erg: Boolean;
Pred	lft. den Vorgänger des Arg. zurück;	Dekl: Pred (x);	Erg: hat denselben Typ wie Arg.;
Succ	lft. den Nachfolger des Arguments zurück;	Dekl: Succ (x);	Erg: hat denselben Typ wie Arg.;

S T R I N G P R O Z E D U R E N U N D - F U N K T I O N E N:

Concat	verbindet eine Folge von Strings und liefert das Erg.;	Dekl: Concat (s1 [,s2,s3,...]: String);	Erg: String;
Copy	lft. einen Teil (cnt Zeichen ab Pos i) eines Strings;	Dekl: Copy (s: String; i, cnt: INTEGER);	Erg: String;
Delete	löscht einen Teil (cnt Zeichen ab Pos i) eines Str.;	Dekl: Delete (VAR s: String; i, cnt: INTEGER);	
Insert	fügt einen Stringteil in einen String (ab Pos i) ein;	Dekl: Insert (source: String; VAR dest: String; i: INTEGER);	
Length	lft. die dyn. Länge des Strings zurück;	Dekl: Length (s: String);	Erg: INTEGER;
Pos	sucht Zeichenfolge innerhalb eines Str. u. lft. ihre Pos.;	Dekl: Pos (substring: String; s: String);	Erg: BYTE;
Str	konvertiert einen numerischen Wert in einen String;	Dekl: Str (x [:width [:decimals]], VAR s: String);	
Val	konvertiert einen String in einen numerischen Wert;	Dekl: Val (s: String; v; VAR code: INTEGER);	

Z E I G E R U N D A D R E S S F U N K T I O N E N:

Addr	lft. die Adresse des angeg. Objekts;	Dekl: Addr (x);	Erg: Pointer;
CSeg	lft. den mom. Wert des CS-Registers;	Dekl: CSeg;	Erg: WORD;
DSeg	lft. den mom. Wert des DS-Registers;	Dekl: DSeg;	Erg: WORD;
Ofs	lft. d. Offsetadr. des ang. Objekts, dh die Adr. innerh. eines Segments.;	Dekl: Ofs (x);	Erg: WORD;
Ptr	konv. 2 Werte für Segment u. Offset in einen Zeigerwert u. lft. d. Erg.;	Dekl: Ptr (seg, ofs: WORD);	Erg: Pointer;
Seg	lft. die Segmentadr. des ang. Obj.;	Dekl: Seg (x);	Erg: WORD;
SPtr	lft. den mom. Wert des SP-Reg.;	Dekl: SPtr;	Erg: WORD;
SSeg	lft. den mom. Wert des SS-Reg.;	Dekl: SSeg;	Erg: WORD;

P R O Z E D U R E N U N D F U N K T I O N E N Z U R D A T E I - B E A R B E I T U N G:

Append	öffnet eine Datei f. d. Anhängen weiterer Dateien;	Dekl: Append (VAR f: Text);
Assign	ordnet einer Datei-Var. eine externe Datei zu;	Dekl: Assign (VAR Name: String);
BlockRead	liest ein od. mehrere Records einer Datei in eine Puffervariable; (result gibt die Anz. der komplett gelesenen Records wieder);	Dekl: BlockRead (VAR f: File; VAR buf; count: WORD; [,result: WORD]);
BlockWrite	schreibt ein od. mehr. Records aus einer Puffervar.in eine Datei;	Dekl: BlockWrite (VAR f:File; VAR buf; count: WORD;[,result:WORD]);
ChDir	wechselt das Standard-Directory;	Dekl: ChDir (s: STRING);
Close	schließt eine Datei;	Dekl: Close (f) (f=DateiVariable);
Eof	prüft ob das Ende einer Datei erreicht ist;	Dekl: Eof(f) / Eof(VAR f: Text); Erg: Boolean;
Eoln	prüft ob die Pos. innerh. einer Dat. auf einem Zeilenende steht;	Dekl: Eoln(f); Eoln(VAR f:Text); Erg: Boolean;
Erase	löscht eine Datei;	Dekl: Erase (f);
FilePos	liefert die mom. Pos. innerhalb einer Datei;	Dekl: FilePos (f); Erg: LONGINT;
FileSize	liefert die Größe einer Datei;	Dekl: FileSize (f); Erg: LONGINT;
.		
.		
GetDir	ermittelt das mom. ges. Directory eines Lw.;	Dekl: GetDir (d: BYTE; VAR s: STRING);
IOResult	liefert den Fehlerstatus der letzten I/O-Operation;	Dekl: IOResult; Erg: WORD;
MkDir	erzeugt ein Subdirectory;	Dekl: MkDir (s: STRING);
Read	liest eine od. mehr. Komponenten aus einer typisierten Datei bzw. einen od. mehrere Werte aus einer Textdatei in die ang. Variablen;	Dekl: Read (f, v1 [,v2,..,vn]); Read ([VAR f: Text;] v1 [,v2,..,vn]);
ReadLn	führt einen Aufruf von READ aus u. springt dann zum Anfang der nächsten Zeile innerhalb der ang. Datei	Dekl: ReadLn ([VAR f: Text;] v1 [,v2,..,vn]);
Rename	gibt einer ext. Diskettendatei einen neün Namen;	Dekl: Rename (f; newname: STRING);
Reset	öffnet eine exist. Datei (vorher ASSIGN);	Dekl: Reset (f [:FILE; recsize: WORD]);
Rewrite	erzeugt und eröffnet eine neü Datei;	Dekl: Rewrite (f[: file; recsize: WORD]);
RmDir	löscht ein leeres Subdirectory;	Dekl: RmDir (s: STRING);
Seek	setzt den Pos.zeiger in einer Datei auf eine best. Komponente;	Dekl: Seek (f; n: LONGINT);

```
SeekEof          prüft ob sich zw. d. mom. Pos. u. d. Ende einer Datei noch lesbare Daten bef.;    Dekl: SeekEof [(VAR f: Text)];
SeekEoln         prft. ob sich zw.d.mom.Pos.u.d.folg. Zeilenende einer Dat.noch lesbare Daten bef.;Dekl: SeekEoln [(VAR f: Text)];
SetTextBuf       weist einer Textdatei-variablen einen Puffer zu;          Dekl: SetTextBuf (VAR f: Text; VAR buf [; size: WORD]);
Truncate         schneidet eine Datei an der mom. Pos. ab;                 Dekl: Truncate (f);
Write            schreibt Daten in eine Datei;       Dekl: Write (f, v1 [,v2,..,vn]); oder Write ([VAR f: Text;] v1 [,v2,..,vn]);
WriteLn          schreibt Daten in eine Datei und schließt mit einem CReturn ab;   Dekl: WriteLn ([VAR f: Text;] v1 [,v2,..,vn]);

V E R S C H I E D E N A R T I G E   S T A N D A R D P R O Z E D U R E N   U N D   F U N K T I O N E N:
FillChar         füllt Speicherber. (Staddr x) ang. Gr.(cnt) BYTEweise mit best. Wert;   Dekl: FillChar (VAR x, cnt: WORD; ch)
Hi               lft. das höherwertige BYTE des Arg.;                                    Dekl: Hi (x), x: Int. od WORD; Erg: BYTE;
Lo               lft. das niederwertige BYTE des Arg.;                                   Dekl: Lo (x), x: Int. od WORD; Erg: BYTE;
Move             kopiert eine Anz. von BYTEs v. einem Speicherber. in einen anderen;     Dekl: Move (VAR source, dest; cnt: WORD);
ParamCount       lft.d. Anz. d. Kommandozeilen-Param. mit denen ein Prg. gestartet wurde;Dekl: ParamCount;        Erg: WORD;
ParamStr         lft. einen Kommandozeilen-Parameter;                                    Dekl: ParamStr (index: WORD);Erg: String;
Random           lft. eine Zufallßahl (mit Randomize initialisieren);                    Dekl: Random [(Range: WORD)];
Randomize        initialisiert den eingebauten Zufallßahlengenerator;                    Dekl: Randomize;
SizeOf           lft. die Gr. des ang. Obj. in BYTEs;                                    Dekl: SizeOf (x);           Erg: WORD;
Swap             vertauscht höherwert. und niederwert. BYTE des Arg.                     Dekl: Swap (x);             Erg: wie Arg.;
UpCase           konvertiert ein Zeichen in einen Großbuchstaben;                        Dekl: UpCase (ch: Char);    Erg: Char;

10) UNIT DOS:
-------------
I N T E R R U P T - P R O Z E D U R E N:
GetIntVec        ermittelt die Adr. auf die ein Intr.-Vektor zeigt; Dekl: GetIntVec (IntNo: BYTE; VAR Vector: Pointer);
Intr             führt einen Software-Intr. aus;                   Dekl: Intr (IntNo: BYTE; VAR Regs: Registers);
MsDos            führt einen DOS-Funktionsaufruf aus;              Dekl: MsDos (VAR Regs: Registers);
SetIntVec        setzt einen Interrupt-Vektor auf eine best. Adr.; Dekl: SetIntVec (IntNo: BYTE; Vector :Pointer);


P R O Z E D U R E N   F U E R   D A T U M   U N D   U H R Z E I T:
GetDate          ermittelt das mom. Kalenderdatum des Systems;             Dekl: GetDate (VAR Year, Month, Day, DayOfWeek: WORD);
GetFTime         erm. Datum/Uhrzeit d. letzten Veränd. einer offenen Datei; Dekl: GetFTime (VAR f; VAR Time: LONGINT)
GetTime          ermittelt die mom. gesetzte Uhrzeit des Systems;          Dekl: GetTime (VAR Hour, Min, Second, Sec100: WORD);
PackTime         konv.Rec. d.Typs DATETIME in Strukt. v. 4 BYTEs f.SETFTIME;Dekl: PackTime (VAR DT: DateTime; VAR Time: LONGINT);
SetDate          setzt das KalenderDatum des Systems;                      Dekl: SetDate (Year, Month, Day, DayOfWeek: WORD);
SetFTime         setzt Modifikationsdatum und Uhrzeit einer offenen Datei; Dekl: SetFTime (VAR f Time: LONGINT);
SetTime          setzt die Uhrzeit des Systems;                            Dekl: SetTime (Hour, minute, Second, Sec100: WORD);
UnpackTime       konv. eine gepackte Struktur von 4 BYTEs in einen Record  Dekl: UnpackTime (Time: LONGINT; VAR DT: DateTime);
                 des Typs DateTime; Derartige Strukturen werden von GetFTime, FindFirst und FindNext erzeugt;
```

S T A T U S F U N K T I O N E N F U E R F D - U N D H D - L A U F W E R K E:

DiskFree — lft. die Anz. d. freien BYTEs auf dem ang. Lw.; Dekl: DiskFree (Drive: WORD); Erg: LONGINT
(0..mom. akt. Lw, 1=A, 2=B,... (gilt auch für DiskSize));

DiskSize — lft. die Ges.Gr. des Datenträgers im ang. Lw.; Dekl: DiskSize (Drive: WORD); Erg: LONGINT

P R O Z E D U R E N Z U R B E A R B E I T U N G V O N D A T E I E I N T R A E G E N:

FindFirst — sucht angeg. bzw. mom. gesetztes Directory nach dem 1. Eintrag ab, der einen best. Dateinamen und festgelegten Attributen entspr.; Dekl: FindFirst (Path: String; Attr: BYTE; VAR S: SearchRec);
Attr.konst: ReadOnly =$01, Hidden =$02, SysFile =$04, VolumeID =$08, Directory =$10, Archive =$20, AnyFile =$3F;

FindNext — setzt eine mit FindFirst begonnene Suche fort; Dekl: FindNext (VAR S: SearchRec);

GetFAttr — ermittelt die Attribute einer Datei; Dekl: GetFAttr (VAR f; VAR Attr: WORD); (Attr. wie oben);

SetFAttr — setzt die Attribute einer Datei; Dekl: SetFAttr (VAR f; Attr: BYTE); (Attr. wie oben);

P R O C E D U R E N U N D F U N K T I O N E N F U E R P R O Z E S S E:

DosExitCode — lft. den Exit-Code eines mit Execute gestarteten Progr; Dekl: DosExitCode; Erg: WORD;

Execute — startet ein Progr., dem zus. Kommandozeilen-Param. übergeben werden können; Dekl: Exec (Path, CmdLine: String);
(Fehler werden in der globalen Variable 'DosError' gespeichert);
Achtung: bei Compil. eines Prgr., d. EXEC/KEEP benutzt muß auf jed. Fall eine Max.Gr. d. Heap festgelegt werden.

Keep — beendet ein Programm und macht es speicherresisdent; Dekl: Keep (ExitCode: WORD)

.
.

11) UNIT Crt:

AssignCrt	ordnet dem Bildschirm eine Textdatei-Variable zu;	Dekl: AssignCrt (VAR f: Text);
ClrEol	löscht sämtl. Zeichen ab d. mom. Pos. d. Crs bis zum rechten Fensterrand;	Dekl: ClrEol;
ClrScr	löscht den ges. Fensterinhalt u. setzt d. Curs. in die l.o. Ecke des Fensters	Dekl: ClrScr;
Delay	hält die Programmausf. für die ang. Anz. von Millisek. an;	Dekl: Delay (ms: WORD);
DelLine	löscht Zeile in d. sich d. Crs bef., schiebt d. unteren Zeilen nach; letzte Zeile leer;	Dekl: DelLine;
GotoXY	Setzt d. Text-Crs auf d. ang. Spalten- Zeilenpos., rel. zur linken, ob. Ecke d. Windows;	Dekl: GotoXY (x, y: BYTE);
HighVideo	setzt d. Zeichenfarbe auf hervorgehoben (Bsp: glob. Var. TextAttr:=LightGray);	Dekl: HighVideo;
InsLine	fügt Leerzeile an d. Cursorpos ein, untere Zeilen werden abwärtsgerollt;	Dekl: InsLine;
KeyPressed	prüft die Tast. u. lft. TRUE zurück wenn eine Taste gedr. wurde, ans. FALSE;	Dekl: KeyPressed;
LowVideo	setzt die Zeichenf. auf halbe "Intensit."; (Bsp: glob. Var. TextAttr:=White);	Dekl: LowVideo;
NormVideo	setzt die Zeichenfarbe auf "normal";	Dekl: NormVideo;
NoSound	schaltet den eingebauten Lautsprecher ab;	Dekl: NoSound;
ReadKey	liest ein Zeichen v. d. Tast., ohne es autom. auf d. Bildschirm außugeben;	Dekl: ReadKey; Erg: Char;
RestoreCrt	setzt wieder den Videomodus, der beim Start des Programms aktiv war;	Dekl: RestoreCrt;
Sound	erzeugt einen Ton im eingebauten Lautsprecher;	Dekl: Sound (Hz: WORD);
TextBackground	setzt die Hintergrundfarbe für folgende Textausgaben;	Dekl: TextBackground (Color: BYTE);

TextColor	setzt die Zeichenfarbe für folgende Textausgaben; (Konst.: (Vg/Hg:) Black =0, blue =1, Green =2, Cyan =3, Red =4, Magenta =5, Brown =6, LightGray =7) (Vg:) DarkGrey=8,Lightblue=9,LightGreen=10,LightCyan=11,LightRed=12,LightMagenta=13,Yellow=14,White=15,Blink=128);	Dekl: TextColor (Color: BYTE);
TextMode	setzt einen (Text-)Videomodus; Konst.:BW40=0,C40=1,BW80=2,C80=3,Mono=7,Last=-1;	Dekl: TextMode (Mode: WORD);
WhereX	lft. d. mom. Spaltenpos.(x) d. Crs rel. zur linken Fensterkante (beginnt mit 1)	Dekl: WhereX; Erg: BYTE;
WhereY	lft. d. mom. Zeilenpos.(y) d. Crs rel. zur oberen Fensterkante (beginnt mit 1);	Dekl: WhereY; Erg: BYTE;
Window	def. ein Textfenster auf d. Screen; (x1,y1: l.o.; x2,y2: r.u.; Crs auf 1,1);	Dekl: Window (x1, y1, x2, y2: BYTE);

12) UNIT Graph:

Arc	zeichnet einen Kreisbogen v. einem Start- zu einem End- winkel u. verw. eine Koord. (x,y) als Kreismittelpkt;	Dekl: Arc (x,y: INTEGER; StartAngle, EndAngle, Radius: WORD);
Bar	zeichnte einen Balken mit d. mom. ges. Farbe u. d. akt. Füllmuster; (x1,y1: links oben, x2,y2: rechts unten);	Dekl: Bar (x1, y1, x2, y2: INTEGER);
Bar3D	zeichnet einen 3-dim. Balken mit der mom. ges. Farbe u. dem akt. Füllmuster; (Depth: räuml. Tiefe, typ.: 25% Breite (x2-x1 div 4); Top: Konst: TopOn (TRUE), TopOff (FALSE) (Deckel zeichnen))	Dekl: Bar3D (x1, y2, x2, y2: INTEGER; Depth: WORD; Top: Boolean);
Circle	zeichnet einen vollst. Kreis mit geg. Rad. um einen Mittelpkt x,y	Dekl: Circle (x, y: INTEGER; Radius: WORD);
ClearDevice	setzt alle Parameter des Grafikpakets auf Standardwerte;	Dekl: ClearDevice;
ClearViewPort	löscht das mom. gesetzte Grafikfenster;	Dekl: ClearViewPort;
CloseGraph	beendet das Grafikpaket;	Dekl: CloseGraph;
DetectGraph	prüft die vorh. Hardware u. best. welcher Grafiktreiber geladen u. welcher Modus gesetzt werden muß; Konst.: Detect=0 (autom. Erkennung), CGA=1, MCGA=2, EGA=3, EGA64=4, EGAMono=5, RESERVED=6, HercMono=7, VGA=9;	Dekl: DetectGraph (VAR GraphDriver, GraphMode: INTEGER);
DrawPoly	zeichnet den Umriß eines Polygons in mom. ges. Linenfarbe und -art;	Dekl: DrawPoly (NuPts: WORD; VAR PolyPoints);
Ellipse	zeichnet einen ellipt. Kreisausschnitt von einem Start zu einem Endwinkel um einen Mittelpunkt (x,y) herum; Dekl: Ellipse (x,y: INTEGER; StartAngle, EndAngle: WORD; XRadius, YRadius: WORD);	
FillPoly	füllt ein Polygon;	Dekl: FillPoly (NuPts: WORD; VAR PolyPoints);
FloodFill	füllt einen umschlossenen Bereich mit dem mom. über SetFillStyle bzw. SetFillPattern gesetzten Muster;	Dekl: FloodFill (x, y, Border:WORD);
GetArcCoords	bestimmt die Koord. des zuletzt gez. Kreisausschnitts;	Dekl: GetArcCoords (VAR AC: ArcCoordsType);
GetAspectRatio	bestimmt das tatsl. Darstellungsverhältnis des benutzten Bilschirms;	Dekl: GetAspRatio (VAR XAsp, YAsp: WORD);
GetBkColor	lft. die mom. ges. Hintergrundfarbe;	Dekl: GetBkColor; Erg: WORD;
GetColor	lft. die mom. ges. Zeichenfarbe zurück;	Dekl: GetColor; Erg: WORD;
GetFillSettings	fragt d. mit SetFillStyle od. SetFillPattern ges. Param. ab;	Dekl: GetFillSettings (VAR FillInfo: FillSettingsType);
GetGraphMode	lft. den mom. ges. Grafikmodus zurück;	Dekl: GetGraphMode; Erg: INTEGER;
GetImage	speichert den Inhalt eines rechteckigen Bilausschnitts bitweise in einer Puffervariablen; Dekl: GetImage (x1, y1, x2, y2: WORD; VAR BitMap);	
GetLineSettings	fragt die durch SetLineStyle gesetzten Parameter ab (Art, Muster, Dicke); Dekl: GetLineSettings (VAR LineInfo: LineSettingsType); Konst: siehe SetLineStyle;	

GetMaxX	lft. die grösstmgl. X-Koord. des mom. akt. Grafikmodus;	Dekl: GetMaxX; Erg: WORD;
GetMaxY	lft. die grösstmgl. Y-Koord. des mom. akt. Grafikmodus;	Dekl: GetMaxY; Erg: WORD;
GetPalette	ermittelt die mom. gesetzte Farbpalette und ihre Größe;	Dekl: GetPalette (VAR Palette: PaletteType);
GetPixel	lft. die Farbe des Pixels auf den angegebenen Koordinaten zurück;	Dekl: GetPixel (x, y: INTEGER); Erg: WORD;
GetTextSettings	fragt d. mit SetTextStyle u. SetTextJustify best. Param. ab;	Dekl: GetTextSettings (VAR TextInfo: TextSettingsType);
	Konst: DefaultFont =0, TriplexFont =1, SmallFont=2, SansSerifFont =3, GothicFonst =4;	
	HorizDir =0, VertDir =1, NormSize =1;	
GetViewSettings	ermittelt die Grenzen des mom. ges. Grafikfensters und die Bed. für das Clipping;	
	Dekl: GetViewSettings (VAR ViewPort: ViewPortType);	
GetX	lft. die mom. X-Position des Grafik-Cursors;	Dekl: GetX; Erg: INTEGER;
GetY	lft. die mom. Y-Position des Grafik-Cursors;	Dekl: GetY; Erg: INTEGER;
GraphErrorMsg	lft. die zum ang. Fehlercode gehörige Meldungals String;	Dekl: GrapherrorMsg (ErrorCode: INTEGER);
.		
.		
GraphResult	lft. den mom. Fehlerstatus des Grafikpakets als INTEGERwert;	Dekl: GraphResult; Erg: INTEGER;
	Konst: grOk =0, grNoInitGraph =-1, grNotDetected =-2, grFileNotfound =-3, grInvalidDriver =-4, grNoLoadMem =-5;	
	grNoScanMem =-6, grNoFloodMem =-7, grFontNoFound =-8, grNoFontMem =-9, grInvalidMode =-10;	
ImageSize	lft. die Anz. v. BYTEs, die zur Speicherung eines best.	Dekl:ImageSize (x1, y1, x2, y2: WORD);
	Ausschnitts mit GetImage gebraucht werden;	
InitGraph	initialisiert das Grafikpaket und schaltet in den Grafikmodus um; (Konst: Siehe DetectGraph)	
	Dekl: InitGraph (VAR GraphDriver: INTEGER; VAR GraphMode: INTEGER; DrivePath: String);	
Line	zeichnet eine Linie von (x1,y1) nach (x2,y2);	Dekl: Line (x1, y1, x2, y2: INTEGER);
LineRel	zeichnet eine Linie rel. zur mom. Pos. des Grafik-Cursors;	Dekl: LineRel (Dx, Dy: INTEGER);
LineTo	zeichnet eine Linie v. d. mom. Pos. d. Grafik-Crs aus zur ang. Koord.;	Dekl: LineTo (x, y: INTEGER);
MoveRel	bewgt. den Grafik-Cursor von seiner mom. Pos. aus um den ang Betrag;	Dekl: MoveRel (Dx, Dy: INTEGER);
MoveTo	setzt den Grafik-Cursor auf den ang. Punkt;	Dekl: MoveTo (x, y: INTEGER);
OutText	gibt einen String ab der mom. Pos. des Grafik-Cursors aus;	Dekl: OutText (TextString: String);
OutTextXY	gibt einen String ab der ang. Pos. aus;	Dekl: OutTextXY (x,y: INTEGER; TStr: String);
PieSlice	zeichnet einen Kreisauschnitt, verbindet seine Enden mit dem Mittelstück und füllt die entstandene Fläche;	
	Dekl: PieSlice (x, y: INTEGER; StAngle, EndAngle, Radius: WORD);	
PutImage	kopiert den Inhalt einer Puffervariablen in einen rechteckigen Ausschnitt des Bildschirms;	
	Dekl: PutImage (x,y: WORD; VAR BitMap; BitBlt: WORD);	
	Konst: NormalPut =0, XorPut =1, OrPut =2, AndPut =3, NotPut =4;	
PutPixel	zeichnet einen einz. Punkt auf den ang. Koord.;	Dekl: PutPixel (x, y,: INTEGER; Color: WORD);
Rectangle	zeichnet ein Rechteck in der mom. ges. Farbe und d. akt. Linienart;	Dekl: Rectangle (x1, y1, x2, y2: INTEGER);
RestoreCrt	setzt den Videomodus der beim Start des Programms aktiv war;	Dekl: RestoreCrt;
RestoreCrtMode	setzt den Videomodus der beim Aufruf von InitGraph aktiv war;	Dekl: RestoreCrtMode;
SetActivePage	setzt eine Bildschirmseite für folgende Ausgaben;	Dekl: SetActivePage (Page: WORD);

```
SetAllPalette     setzt sämtl. Einträge einer Farbpalette neu;                          Dekl: SetAllPalette (VAR Palette);
                  Konst: Black=0, blue =1, Green =2, Cyan =3, Red=4, Magenta =5, Brown =6, LightGrey =7, DarkGrey =8, Lightblue =9,
                         LightGreen =10, LightCyan =11, LightRed =12, LightMagenty =13, Yellow =14, White =15, MaxColors =15;
SetBkColor        setzt die Hintergrundfarbe für folgende Zeichenaktionen;              Dekl: SetBkColor(Color: WORD);
SetColor          setzt d. Zeichenfarbe f. folg. Zeichenakt.(benutzt akt. Farbpalette); Dekl: SetColor (Color: WORD);
SetFillPattern    setzt ein benutzerdefiniertes Füll-Muster;      Dekl: SetFillPattern (Pattern: FillPatternType); Col: WORD);
SetFillStyle      setzt ein vordef. Füllmuster;                                         Dekl: SetFillStyle (Pattern ,Col: WORD);
                  Konst: EmptyFill =0, SolidFill =1, LineFill =2, LtSlashFill =3, SlashFill =4, BkSlshFill =5, LtBkSlashFill =6;
                         HatchFill =7, XHatchFill =8, InterleaveFill =9, WideDotFill =10, CloseDotFill =11, UserFill =12;
SetGraphMode      setzt einen Grafikmodus und löscht den Bildschirm;                    Dekl: SetGraphMode (Mode: INTEGER);
                  Konst.: EGALo, EGAHi, EGAMonoHi, HercMonoHi;
SetLineStyle      setzt Breite und Muster für nachfolgende Linien;                      Dekl: SetLineStyle (LnSt,Muster,Dicke: WORD);
                  Konst.:SolidLn =0, DottedLn =1, CenterLn =2, DashedLn =3, UserBitLn =4; NormWidth =1; ThickWidth =3
SetPalette        setzt einen Eintrag in einer Farb-Palette;                            Dekl: SetPalette (ColNum: WORD; Color:BYTE);
SetTextJustify    setzt Param. f. d. Justierung v. Textausg. via OutText und OutTextXY;
                  Dekl: SetTextJustify (Horiz, Vert: WORD); Konst.: LeftText, CenterText, RightText, BottomText, TopText;
SetTextStyle      setzt einen Zeichensatz, den Stil und den Vergrösserungsfaktor;
                  Dekl: SetTextStyle (Font, Direction,: WORD; CharSize: WORD); (Fonts: siehe GetTextSettings);
SetViewPort       setzt ein Zeichenfenster in der Grafik;  Konst: ClipOn, ClipOff;
                  Dekl: SetViewPort (x1, y1, x2, y2: WORD; Clip: Boolean); (x1, y1: links oben; x2, y2: rechts unten);
SetVisualPage     bringt eine von mehreren Grafikseiten auf den Bildschirm;             Dekl: SetVisualPage (Page: WORD);
TextHeight        lft. die Höhe eines Strings (in Punkten);                             Dekl: TextHeight (TxtStrg: STRING); Erg:WORD;
TextWidth         lft. die Breite eines Strings (in Punkten);                           Dekl: TextWidth (TxtStrg: STRING); Erg: WORD;

13) Vordefinierte Typen:
------------------------
SearchRec = RECORD           ! FillSettingsType = RECORD             ! TextSettingsType = RECORD ! ViewPortType = RECORD
   Fill: ARRAY[1..2] OF BYTE; !   Pattern: WORD;                     !    Font     : WORD;       !   x1, y1, x2, y2: WORD;
   Attr: BYTE;               !   Color  : ARRAY [1..8] OF SHORTINT  !    Direction: WORD;       !   Clip          : BOOLEAN;
   Time: LONGINT;            ! END;                                  !    ChartSize: WORD;       ! END;
   Size: LONGINT;            !                                       !    Horiz    : WORD;       !
   Name: STRING [12];        !                                       !    Vert     : WORD;       !
END;                         ! FillPatternType = ARRAY [1..8] OF BYTE ! END;                      !

LineSettingsType = RECORD ! PalleteType = RECORD ! ArcCoordsType = RECORD              ! Register = RECORD
  LineStyle: WORD;        !   Size:   WORD;      !   X, Y      : INTEGER; (Mittelpunkt) !  CASE INTEGER OF
  Pattern  : WORD;        !   Colors: WORD;      !   Xs, Ys,              (Startpunkt)  !  0: (AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags:WORD);
  Thickness: WORD;        ! END;                 !   Xend, Yend: WORD;    (Endpunkt)    !  1: (AL,AZH,BL,BH,CL,CH,DL,DH: BYTE);
END;                      !                      ! END;                                 ! END;
```

LISTENVERARBEITUNG MIT TURBO PASCAL 4.0

TGM_96: LISTEN.TXT, TGM_97:LISTEN.ARC
Ernst Huber, N86d

1) Einführung:

Innerhalb vieler Programme fallen wärend der Laufzeit Daten an, deren Menge von Parametern abhängt, die vom Benutzer oft willkürlich festgelegt werden können/dürfen. Folglich ist die Menge der Daten (ich möchte im weiteren von Datensätzen oder Records sprechen) hochgradig variabel.

Viele unerfahrene Programmierer verwenden zur Speicherung solcher Datenmengen teils aus Unwissenheit, teils aus Bequemlichkeit, teils weil sie es aus einer anderen Programmiersprache (BASIC) gewohnt sind, Arrays.

Einziger Vorteil dieser Methode ist, daß man auf jeden beliebigen Datensatz (Record) unter Angabe einer Nummer komfortabel und schnell zugreifen kann. Damit wären wir aber schon bei den Nachteilen.

a) Die Anzahl der Records, die wärend der Laufzeit des Programmes anfallen, muß dem Programmierer von vornherein bekannt sein, oder er dimensioniert das Datenarray so groß, daß die für jeden gültigen Eingabefall auftretende Datenmenge in das Array paßt (Worst-case Dimensionierung). Damit wird aber wiederum Speicherplatz belegt, der vielleicht gar nie benötigt und anderweitig dringender gebraucht wird (z.B. zusätzliche Variable/Arrays im Datensegment - siehe Punkt b).

b) Das Datensegment eines Programmes kann max. 64 kB (65535 Bytes) groß sein. Innerhalb dieses Datensegmentes werden in Turbo Pascal jedoch sämtliche vom Programm benutzten Variablen abgelegt, also auch unser Datenarray. Oder anders ausgedrückt: das Array kann theoretisch nur maximal 64 kB groß sein. Dies bedeutet, daß die Anzahl der Arrayelemente (Records) von vornherein beschränkt ist.

Zur besseren Veranschaulichung möchte ich nun mit einem Beispiel beginnen, für daß ich im weiteren mehrere Lösungsansätze bringen werde.

Beispiel: Erstellen eines Programmes zur Verwaltung einer Ersatzteilliste;

Aufgabenstellung: Mittels Turbo Pascal 4.0 soll ein (für uns imaginäres) Programm geschrieben werden, anhand dessen es möglich ist, eine Ersatzteilliste bequem und rasch zu verwalten (komfortables Arbeiten, schneller Zugriff auf die einzelne Datensätze). Die Ersatzteilliste (oder zumindest immer ein großer Teil davon) soll daher aus Geschwindigkeitsgründen während des Programmablaufes im Speicher gehalten werden.

Zu jedem Ersatzteil gibt es charakterische Daten zu speichern (Bezeich- nung, Artikelnummer, Anzahl, Preis);

Unser Problem ist jetzt der Aufbau einer geeigneten Datenstruktur und die dazugehörige Speicherverwaltung.

Beginnen wir mit der programmiertechnisch naivsten Methode (siehe nebenstehendes Programm-Listing):

Bedenkt man, daß in einem ansprechenden Programm sicherlich noch weitere Variable und kleine Arrays hinzukommen werden, so wird es im Datensegment recht eng zugehen (außerdem sind 500 Datensätze wirklich nicht viel - höchstens zum Tippen).

Eine etwas elegantere Lösung bietet folgendes Beispiel (jedoch keine Verbesserung im Bezug auf den Platzbedarf im Datensegment, siehe nächste Seite):

```
PROGRAM Verwaltel;
{ Zu jedem der oben genannten charakteristischen Daten wird ein Array
  definiert, jedem Ersatzteil eine Nummer zugeordnet. Weiters wird ein
  Zugriff auf einen Ersatzteil-Datensatz demonstriert, der die spezifischen
  Daten zur Anzeige bringt (Voraussetzung ist, daß ein Heinzelmännchen
  bereits alle erforderlichen Datensätze eingegeben hat). }

USES CRT;

CONST MaxDaten = 500;              { Max. 500 Datensätze pro Ersatzteilliste }

VAR Bezeichnung: ARRAY [1..MaxDaten] OF STRING[100];{ 500*100= 50000 Bytes }
    Artikelnr  : ARRAY [1..MaxDaten] OF LONGINT;    { 500*4  =  2000 Bytes }
    Anzahl     : ARRAY [1..MaxDaten] OF WORD;       { 500*2  =  1000 Bytes }
    Preis      : ARRAY [1..MaxDaten] OF WORD;       { 500*2  =  1000 Bytes }
    cnt        : WORD;                              {             2 Bytes }
                                                    { Gesamt:  54002 Bytes }
{ Theoretisch können also max. 65531 DIV 108 = 606 Ersatzteile/Liste erfaßt
  werden }

BEGIN
  ...
  < S O U R C E >
  ...
  WriteLn ('Zugriff auf den Ersatzteil-Datensatz mit Nr.: ',cnt);
  WriteLn ('Ersatzteil   : ', Bezeichnung [cnt]);
  WriteLn ('Artikelnummer: ', Artikelnr [cnt]);
  WriteLn ('Anzahl       : ', Anzahl [cnt]);
  WriteLn ('Preis        : ', Preis [cnt]);
  ...
  < S O U R C E >
  ...
END. (* PROGRAM Verwaltel *)
```

```
PROGRAM Verwalte2;
{ Für jeden Ersatzteil wird nun ein RECORD definiert, dies bringt zwar keine
  Platzersparnis, ist jedoch bei weitem übersichtlicher. Weiters wird ein
  Zugriff auf einen Datensatz gezeigt. }

USES CRT;

CONST MaxDaten = 500;  { Max. 500 Datensätze pro Ersatzteilliste }

TYPE Ersatzteil = RECORD   { Definition unseres Ersatzteilrecords}
       Bezeichnung: STRING[100];                    {100  Bytes }
       Artikelnr  : LONGINT;                        {  4  Bytes }
       Anzahl     : WORD;                           {  2  Bytes }
       Preis      : WORD;                           {  2  Bytes }
     END; (* RECORD Ersatzteil *)           { Gesamt: 108  Bytes/RECORD }

VAR ErsatzListe: ARRAY [1..MaxDaten] OF Ersatzteil; { 500*108= 54000 Bytes }
    cnt        : WORD;                              {              2 Bytes }
                                                    { Gesamt:  54002 Bytes }
{ Theoretisch können also max. 65531 DIV 108 = 606 Ersatzteile/Liste erfaßt
  werden  }

BEGIN
  ...
  < S O U R C E >
  ...
  WITH ErsatzListe [cnt] DO BEGIN
       WriteLn ('Zugriff auf den Ersatzteil-Datensatz mit Nr: ',cnt);
       WriteLn ('Ersatzteil     : ', Bezeichnung);
       WriteLn ('Artikelnummer  : ', Artikelnr);
       WriteLn ('Anzahl         : ', Anzahl);
       WriteLn ('Preis          : ', Preis);
  END; (* WITH ErsatzListe [cnt] *)
  ...
  < S O U R C E >
  ...
END. (* PROGRAM Verwalte2 *)
```

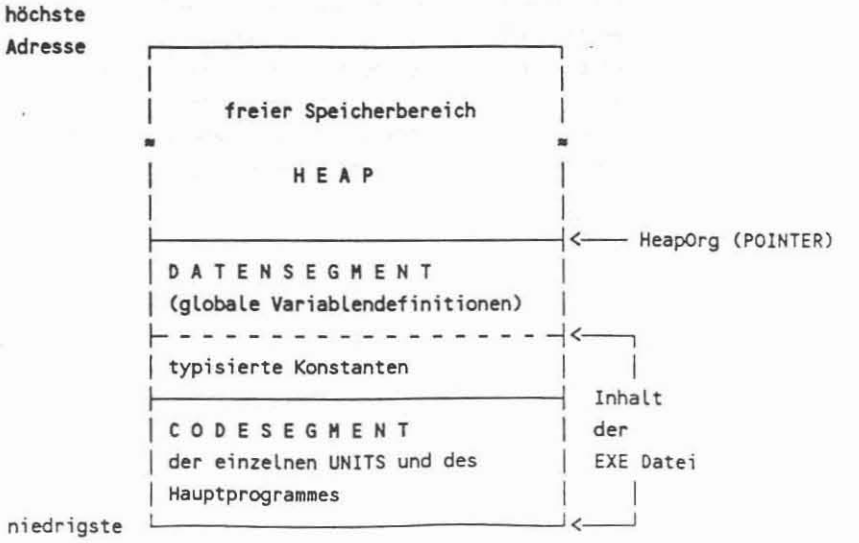
Die meisten PC XT/AT/... sind aber heute bereits mit mindestens 640 kB Hauptspeicher ausgestattet (welche unter MS-DOS/PC-DOS maximal ohne Tricks verwaltet werden können). Unter Berücksichtigung von DOS, eventuell residenter Utilities sowie unseres laufenden Verwaltungsprogrammes bleiben also immer noch 400 kB (und mehr) an freiem Hauptspeicher über, der nur darauf wartet, mit unseren Datenbits und -bytes gefüllt zu werden.

Ideal wäre also eine Möglichkeit, diesen freien Speicherplatz während des Programmablaufes belegen und wieder freigeben zu können. Diese existiert tatsächlich (viele haben sich's doch bereits gedacht -- es ist wie bei einer dieser Hollywood-Komödien mit Happy End -- oder!??), man spricht dann von einer dynamischen Speicherverwaltung. Der freie Speicherbereich wird Heap (dt. Haufen, Menge) genannt. Diese dynamische Speicherverwaltung stellt Turbo Pascal in hervorragendem Maße zur Verfügung, und mit der wollen wir uns nun näher befassen.

2) Dynamische Speicherverwaltung:

Im letzten Kapitel haben wir den Begriff 'Heap' kennengelernt, schauen wir uns nun anhand einer Skizze näher an, was es damit auf sich hat.

Skizze des Speicherschemas für ein compiliertes Turbo Pascal 4.0 Programm (stark vereinfacht):



Am Heap können nun einzelne Bytes oder zusammenhängende Speicherbereiche mittels der unten angeführten Prozeduren reserviert werden. Innerhalb eines reservierten Speicherbereiches können nun Daten abgelegt werden, man muß dazu nur seine Startadresse kennen (und eventuell die Größe, mehr dazu später). Diese Startadresse wird in einer Variablen gespeichert, man spricht hier von einer Zeigervariablen oder englisch POINTER.

Turbo Pascal kennt nun zwei Möglichkeiten der dynamischen Speicherverwaltung: (auf eine dritte mit MARK/RELEASE möchte ich hier nicht näher eingehen, der interessierte Leser sei auf das Turbo Pascal 4.0 Referenzhandbuch verwiesen).

a) Speicheranforderung mit: NEW (p: POINTER);
   Speicherfreigabe mit: DISPOSE (p: POINTER);
b) Speicheranforderung mit: GetMem (p: POINTER; Size: WORD);
   Speicherfreigabe mit :FreeMem (p: POINTER; Size: WORD);

(Die Größe einer dynamischen Variablen ist wiederum auf 64 kB beschränkt, dies stellt jedoch im weiteren kein wirkliches Problem dar.)

Die erste Art a) der dynamischen Speicherverwaltung wird verwendet, wenn die Größe des zu reservierenden Bereiches im Heap von vornherein festeht, hingegen kann bei b) die Größe des zu reservierenden Speicherbereiches während der Laufzeit festgelegt werden (also selbst wieder variabel sein).

Folgende Prozeduren/Funktionen und Variablen zur dynamischen Speicherverwaltung werden von Turbo Pascal 4.0 aus vordefiniert.

a) Prozeduren/Funktionen:

Dispose
: gibt den Speicherplatz einer dynamischen Variablen frei;
Dekl: Dispose (VAR p: POINTER);

FreeMem
: gibt einen dynamisch belegten Speicher bestimmter Größe frei;
Dekl: FreeMem (VAR p: POINTER; size: WORD);

GetMem
: belegt Speicher bestimmter Größe für eine dynamische Variable, setzt einen Zeiger auf den Anfang dieses Bereiches;
Dekl: GetMem (VAR p: POINTER; size: WORD);

Mark
: hält den Zustand des Heaps in einer Zeigervariablen fest;
Dekl: Mark (VAR p: POINTER);

MaxAvail
: liefert die Größe des größten freien Heap-Bereiches, ermittelt, wieviele Bytes momentan max. für eine dyn. Var. belegt werden können;
Dekl: MaxAvail; Erg: LONGINT;

MemAvail
: liefert die Gesamtzahl der freien Bytes auf dem Heap;
Dekl: MemAvail; Erg: LONGINT;

New
: erzeugt eine dynamische Variable, setzt Zeiger auf den Anfang des durch diese Variable belegten Bereiches;
Dekl: New (VAR p: POINTER);

Release
: setzt den Heap auf den mit MARK festgehaltenen Zustand zurück;
Dekl: Release (VAR p: POINTER);

b) Variable (hier möchte ich nur die meiner Ansicht nach wichtigsten anführen):

HeapOrg:
: POINTER Zeiger auf die Startadresse des Heapbereiches;

HeapPtr:
: POINTER Zeiger auf die Endadresse des belegten Heapbereiches;

FreePtr:
: POINTER Zeiger auf die Endadresse des noch freien Heapbereiches;

Anmerkung: mittels RELEASE (HeapOrg) kann jederzeit der gesamte Heapbereich freigegeben werden, jedoch Vorsicht im Zusammenhang mit der UNIT GRAPH, hier kann es zu so manchen Problemen kommen;

An dieser Stelle möchte ich eine weitere Lösung zu dem oben angegeben Beispiel bringen. Der folgende Programmansatz stellt bereits eine recht gute Lösung des obigen Problems dar, kann aber noch verbessert werden (die einzelnen Datensätze werden zwar dynamisch gespeichert, jedoch muß noch immer die max. Anzahl der Ersatzteile/Liste vom Programmierer festgelegt werden).

```
PROGRAM Verwalte3;
{ Für jeden Ersatzteil wird nun ein RECORD definiert, weiters wird ein
  ARRAY OF POINTER TO ERSATZTEIL definiert, das heißt jedes Arrayelement
  zeigt auf einen Speicherbereich im Heap, in dem ein Datensatz gespeichert
  wird. Gezeigt wird nun sowohl die Belegung vom Heap, das Ein- und Auslesen
  eines Datensatzes sowie ein anschließendes Freigeben des belegten
  Speicherplatzes für einen Datensatz. }

USES CRT;

CONST MaxDaten = 500;  { Max. 500 Datensätze pro Ersatzteilliste }

TYPE Ersatzteil = RECORD
       Bezeichnung: STRING[100];                    {100  Bytes }
       Artikelnr  : LONGINT;                        {  4  Bytes }
       Anzahl     : WORD;                           {  2  Bytes }
       Preis      : WORD;                           {  2  Bytes }
     END; (* RECORD Ersatzteil *)          { Gesamt: 108  Bytes/RECORD }
```

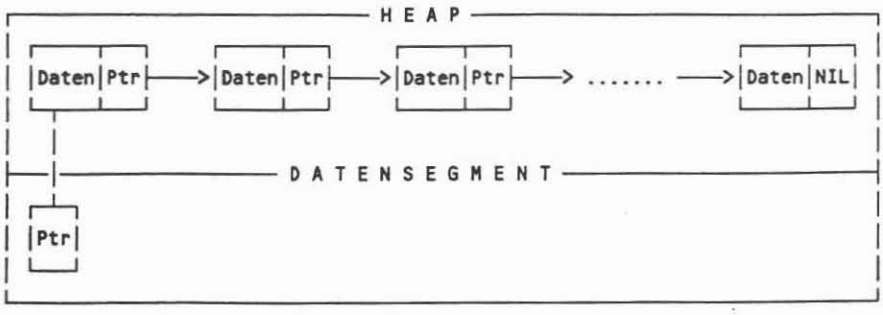
```
VAR ErsatzListe: ARRAY [1..MaxDaten] OF ^Ersatzteil; { 500*4 = 2000 Bytes }
    cnt        : WORD;                               {           2 Bytes }
    Ende       : CHAR;                               {           1 Byte  }
                                                { Datensegment: Gesamt: 2003 Bytes }
                                                { Heapbereich : Gesamt:54000 Bytes }
{ Max. können also 65531 DIV 4 = 16382 Ersatzteile/Liste erfaßt werden; dazu
  müßte man jedoch einen freien Heapbereich von 16382*108 = 1.769.337 Bytes
  haben, (keine Sorge, DOS schafft sowieso nur 640 kB) - und unser heiliges
  Datensegment wäre wieder voll. }

BEGIN
  ...
  < S O U R C E >
  ...
  REPEAT
    INC (cnt);
    NEW (ErsatzListe[cnt]);          (* Belegt einen 108 Byte großen     *)
    WITH ErsatzListe[cnt]^ DO BEGIN  (* Bereich und speichert die St.adr *)
         WriteLn ('Einlesen eines Ersatzteil Datensatz mit Nr: ',cnt);
         Write ('Ersatzteil     : '); ReadLn (Bezeichnung);
         Write ('Artikelnummer  : '); ReadLn (Artikelnr);
         Write ('Anzahl         : '); ReadLn (Anzahl);
         Write ('Preis          : '); ReadLn (Preis);
    END; (* WITH ErsatzListe [cnt] *)
    WriteLn; WriteLn ('Weiter n/Taste : '); Ende:=ReadKey;
    Ende:=UpCase(Ende);
  UNTIL ((cnt >= MaxDaten) OR (Ende = 'N'));
  ...
  < S O U R C E >
  ...
  REPEAT
    INC (cnt);
    WITH ErsatzListe[cnt]^ DO BEGIN
         WriteLn ('Zugriffs auf den Ersatzteil Datensatz mit Nr: ',cnt)
         WriteLn ('Ersatzteil     : ', Bezeichnung);
         WriteLn ('Artikelnummer  : ', Artikelnr);
         WriteLn ('Anzahl         : ', Anzahl);
         WriteLn ('Preis          : ', Preis);
    END; (* WITH ErsatzListe [cnt]^ *)
    WriteLn ('Weiter n/Taste : '); Ende:=ReadKey;
    Ende:=UpCase(Ende);
    DISPOSE (ErsatzListe[cnt]);
  UNTIL ((cnt >= MaxDaten) OR (Ende = 'N'));
  ...
  < S O U R C E >
  ...
END. (* PROGRAM Verwalte3 *)
```

3) Listen und andere dynamische Datenstrukturen:
Mit dem vorhergehenden Programmansatz (bzw. mit der erstellten Datenstruktur) ist es gelungen, den Platzbedarf im Datensegment zur Speicherung von 500 Elementen erheblich zu reduzieren. Im Datensegment wird nur noch ein POINTER-ARRAY gespeichert (500*4 = 2000 Bytes), die eigentlichen Daten (Records) werden im Heap abgelegt, der hierfür genügend freien Speicherplatz bereitstellt. Optimal wäre es nun, wenn es gelänge, selbst dieses ARRAY bzw. die Information, die dieses beinhaltet, ebenfalls im Heap unterzubringen.

Eine Idee, mit der wir uns jetzt näher befassen werden (und die uns schließlich zum Ziel führen wird) ist die folgende:

Im letzten Beispiel haben wir ein Array bestehend aus Pointern auf eine Datenstruktur definiert, das heißt jeweils ein Pointer zeigt auf einen Datensatz, der im Heap gespeichert wird. Was passiert nun, wenn man in jedem Record (der ja selbst aus Variablen besteht) eine Pointervariable unterbringt, die auf den jeweils nächsten Record zeigt.

Dazu folgenden Skizze (siehe nächste Seite):

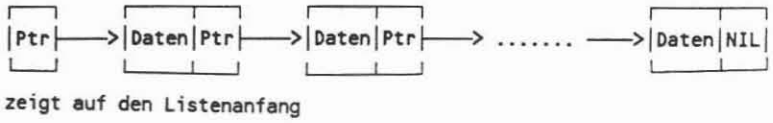


Die einzige Information, die man sich nun im Hauptprogramm mit Hilfe einer Variablen (und damit im Datensegment) merken muß, ist ein Pointer auf das erste Element dieser Recordfolge. Datenstrukturen dieser Art heißen in der Fachsprache Listen, anstatt von Elementen (Datensätze, Records) spricht man auch von Knoten (engl. Nodes).

3.1) Die einfach verkettete Liste:

Eine einfach verkettete Liste besteht aus einer Folge von Knoten. Jeder Knoten beinhaltet einen Pointer auf seinen Nachfolger in der Liste. Am Ende der Liste hat dieser Pointer den Wert NIL (Not-In-List). Die Datenstruktur, die wir uns vorher erarbeitet haben, ist also eine einfach verkettete Liste.

Graphische Darstellung einer einfach verketteten Liste:



zeigt auf den Listenanfang

Das Programmbeispiel sieht dann folgendermaßen aus:

```
PROGRAM Verwalte4;
{ Für jeden Ersatzteil wird wiederum ein Record definiert, der sowohl die
  üblichen Daten als auch einen Zeiger auf das nächste Listenelement
  enthält; gezeigt wird nun sowohl die Belegung vom Heap, das Ein- und
  Auslesen eines Datensatzes sowie ein anschließendes Freigeben des belegten
  Speicherplatzes für einen Datensatz. }


USES CRT;

TYPE Liste = POINTER ^Ersatzteil;
     Ersatzteil = RECORD
       Bezeichnung: STRING[100];                    {100  Bytes }
       Artikelnr  : LONGINT;                        {  4  Bytes }
       Anzahl     : WORD;                           {  2  Bytes }
       Preis      : WORD;                           {  2  Bytes }
       Nachfolger : Liste;                          {  4  Bytes }
     END; (* RECORD Ersatzteil *)          { Gesamt: 112  Bytes/RECORD }

VAR z1, z2, ListenAnfang: Liste;            (* z1, z2 = Hilfsvariable *)
    cnt                 : WORD;
    Ende                : CHAR;

{ Maximal können HeapSize DIV 112 = ??? Ersatzteile/Liste erfaßt werden }

BEGIN
  ...
  < S O U R C E >
  ...
  RELEASE (HeapOrg);        (* löschen des gesamten Heap                 *)
  NEW (ListeAnfang);        (* Platz bel. für den 1. Knoten in der Liste  *)
  z1:=ListenAnfang; cnt:=0; (* Adr. des reservierten Bereichs in Hilfsvar.*)
  REPEAT
    INC (cnt);
    WriteLn ('Einlesen des Ersatzteil Datensatz mit Nr: ',cnt);
    Write ('Ersatzteil     : '); ReadLn (z1^.Bezeichnung);
    Write ('Artikelnummer  : '); ReadLn (z1^.Artikelnr);
    Write ('Anzahl         : '); ReadLn (z1^.Anzahl);
    Write ('Preis          : '); ReadLn (z1^.Preis);
    WriteLn; WriteLn ('Weiter n/Taste : '); Ende:=ReadKey;
```

```
  Ende:=UpCase(Ende);
  IF (Ende <> 'N') THEN BEGIN   (* Wenn eine weitere Eingabe erwünscht, *)
     NEW (z2);                  (* dann über z2 Platz für den nächsten  *)
     z1^.Nachfolger:=z2;        (* Record belegen und dessen Startadr.  *)
     z1:=z2;                    (* als Nachfolger überweisen; nachher   *)
  END;                          (* wird der neue Record als aktueller   *)
                                (* Datensatz gesetzt                    *)
UNTIL ((cnt >= MaxDaten) OR (Ende = 'N'));
z1^.Nachfolger:=NIL;   (* letzter Nachfolger in der Liste muß NIL sein *)
...
< S O U R C E >
...
cnt:=0;
REPEAT
  z1:=ListenAnfang;                (* Startadresse des 1.Records Laden *)
  INC (cnt);
  WriteLn ('Zugriff auf den Ersatzteil Datensatz mit Nr: ',cnt)
  WriteLn ('Ersatzteil     : ', z1^.Bezeichnung);
  WriteLn ('Artikelnummer  : ', z1^.Artikelnr);
  WriteLn ('Anzahl         : ', z1^.Anzahl);
  WriteLn ('Preis          : ', z1^.Preis);
  WriteLn ('Weiter n/Taste : '); Ende:=ReadKey;
  Ende:=UpCase(Ende);
  IF (Ende <> 'N') THEN BEGIN       (* wenn kein Ende erwünscht, dann   *)
      ListenAnfang:=z1^.Nachfolger; (* neuer Listenanf.= nächster Knoten*)
      DISPOSE (z1);                 (* löschen des alten Knotens        *)
  END;
 UNTIL ((ListenAnfang = NIL) OR (Ende = 'N')); (* bis Listenende od. Ende*)
 ...
 < S O U R C E >
 ...
END. (* PROGRAM Verwalte4 *)
```

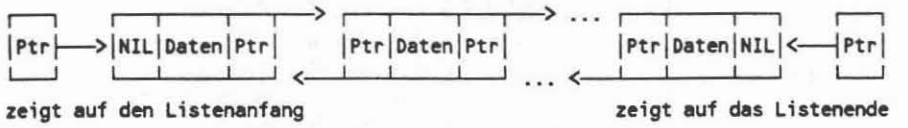
Ein Nachteil dieser Speichermethode ist, daß man, um z.B. im Extremfall an das letzte Listenelement heranzukommen, die gesamte vorhergehende Liste durchlaufen muß.

Die Vorteile wären: keine Festlegung der max. Anzahl von Listenelementen durch den Programmierer un das Datensegment eines Programmes bleibt (fast) unbelegt.

3.2) Die doppelt verkettete Liste:

Eine doppelt verkettete Liste besteht aus Knoten, die jeweils einen Zeiger auf den Vorgänger in der Liste als auch einen Zeiger auf den Nachfolger in der Liste beinhalten. Der Vorgänger des ersten Knotens sowie der Nachfolger des letzten Knotens hat jeweils den Wert NIL.

Graphische Darstellung einer doppelt verkettete Liste:



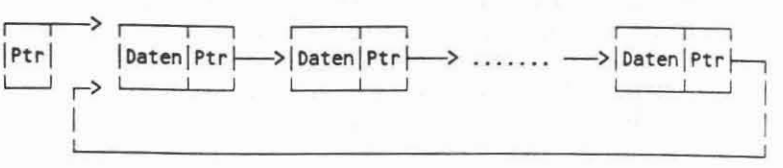
Vorteil: Man kann sowohl vom Listenanfang, als auch vom Listenende aus linear zu suchen beginnen (bzw. von jedem Knoten innerhalb der Liste).

Nachteil: Höherer Platzbedarf (ein Pointer/Node mehr gegenüber einer einfach verketten Liste (bei 5000 Nodes sind dies immerhin ca. 20 kB). Dies wirkt sich vor allem bei Listen mit kleinen Datenfeldern pro Knoten stark aus (z.B. LONGINT Listen).

3.3) Die zyklische Liste:

Der Unterschied zu den beiden vorigen Listenarten besteht hierin, daß der Zeiger des letzten Knotens nicht gleich NIL ist, sondern auf das erste Listenelement zeigt (bzw. auch der Zeiger auf den Vorgänger bei doppelt verketteten zyklischen Listen auf das letzte Listenelement zeigt). Für solche Listen ist auch die Bezeichnung Datenring gebräuchlich.

Graphische Darstellung einer einfach verketteten zyklischen Liste:



3.4) Der binäre Baum:

Ein Baum besteht im Unterschied zu einer Liste aus Knoten, die jeweils mehr als einen Nachfolger haben können. Beim binären Baum besitzt jeder Knoten zwei Nachfolger, das heißt er enthält zwei Zeiger. Am Ende des Baumes haben diese Zeiger den Wert NIL.

4) Die UNIT List_Man(anger):

In diesem Kapitel möchte ich eine selbstgeschriebene UNIT vorstellen, mit der es möglich ist, die Verwaltung von mehreren Listen gleichzeitig auf recht einfache Weise zu gestalten. Dabei wurden für manche Prozeduren bewußt Namen verwendet, wie sie bei rein listenverarbeitenden Programmiersprachen vorkommen (z.B. aus LOGO - das meiner Meinung nach zu Unrecht den Ruf einer Kinderprogrammiersprache genießt).

Grundsätzlich können mittels der UNIT List_Man (doppelt verkettete) Listen mit beliebigen Datenstrukturen für einen Knoten angelegt werden. Einzige Bedingung ist nur, daß eine Variable vom Knotentyp im Hauptprogramm bzw. in der aufrufenden Routine erzeugt wird (mehrer Knotentypen/Liste sind im Normalfall unzulässig, mit einigen Tricks kann auch diese Schranke umgangen werden). Über diese Variable findet dann jegliche Kommunikation zwischen dem Hauptprogramm bzw. der aufrufenden Routine und der Listenverarbeitung statt, das heißt Daten werden über diese Variable in die Liste übergeben und abgeholt.

Weiters m u ß u n b e d i n g t V O R dem Anlegen einer Liste die Größe eines Listenknotens bestimmt und das Ergebnis über die vordefinierte Variable 'ObjectSize' der UNIT List_Man mitgeteilt werden. Es empfiehlt sich die UNIT nach den Systemunits (CRT, DOS, PRINTER, GRAPH) und vor den selbstgeschriebenen Units über USES in das Programm aufzunehmen.

4.1) Vordefinierte (öffentliche) Variable der UNIT List_Man:

```
VAR CurrentNo, LastNo, FirstNo, ObjectSize, IORes: WORD;
    NotExist                                     : BOOLEAN;
    SysPtr                                       : POINTER;
    Current                                      : List;
    ActiveListNo, TopOfList                      : BYTE;
```

CurrentNo: WORD;
Nummer des gerade aktuellen Listenelementes;
FirstNo: WORD;
Nummer des ersten Listenelementes (Numerierung beginn mit 1);
LastNo: WORD;
Nummer des letzten Listenelementes;
IORes: WORD;
Fehlerstatus der letzten IO-Operation bei Diskzugriffe (SaveList, LoadList, KillDisk);
ObjectSize WORD
In dieser Variablen erwartet sich die UNIT die Größe eines Knotens in Byte, aus dem die Liste zusammengesetzt werden soll. Eine Bestimmung der Knotengröße ist immer vor der Verwendung des ML Befehls notwendig (also auch vor LoadList);
NotExist: BOOLEAN;
Diese Variable wird auf TRUE gesetzt, wenn eine aufgerufene Liste nicht aktiviert werden kann (SetActiveList nicht erfolgreich) bzw. keine Liste aktiviert ist.
SysPtr: POINTER;
In diesem Zeiger wird der Zustand des Heap vor der Verwendung der UNIT List_Man (automatisch) festgehalten.
Current: List;
Zeiger auf den gerade aktuellen Knoten;
ActiveListNo: BYTE;
Beinhaltet die Nummer der gerade aktiven Liste;
TopOfList: BYTE;
Gibt die Nummer der letzten Liste im Liststack an.

4.2) Vordefinierte (öffentliche) Prozeduren und Funktionen der UNIT List_Man:

```
PROCEDURE SetFirstNode;
    (* Setzt erstes Listenel als akt.El. *)
PROCEDURE SetLastNode;
    (* Setzt letztes Listenel als akt.El.*)
FUNCTION Next: List;
    (* lft Pointer auf das naechste El. *)
FUNCTION Prev: List;
    (* lft Pointer auf das vorige El *)
FUNCTION ListSize: LONGINT;
    (* lft Listengroesze in Bytes *)
FUNCTION DataSize: LONGINT;
    (* lft Bytes gebraucht fuer Daten *)
FUNCTION ListName: STR08;
    (* lft Listennamen *)
PROCEDURE SetActiveList (Name: STR08);
    (* Setzt ein Liste auf aktiv *)
PROCEDURE ML (VAR Data; Name: STR08);
    (* Legt eine Liste an *)
PROCEDURE KillMem;
    (* Loescht komplette Liste *)
PROCEDURE LPut (VAR Data);
        (* fuegt ein El. a.d. Listenende *)
PROCEDURE FPut (VAR Data);
        (* fuegt ein El. a.d. Listenanf.
        *)
PROCEDURE InsCur (VAR Data);
        (* Efg eines El nach akt. Posi-
        tion *)
PROCEDURE DelCur;
    (* loescht El. an akt. Position *)
PROCEDURE GetNext (VAR Data);
    (* gibt das naechst El. zurueck *)
PROCEDURE GetPrev (VAR Data);
    (* gibt das vorherg. El. zurueck *)
PROCEDURE GetData (VAR Data);
    (* gibt das akt El. zurueck *)
PROCEDURE PutData (VAR Data);
    (* überschreibt den akt. Datensatz *)
PROCEDURE SeekNo (No: WORD);
    (* Sucht das Listenel. mit No *)
PROCEDURE SaveList (VAR Buf; FileName: STR12);
    (* Speichern auf akt. Disk *)
PROCEDURE LoadList (VAR Buf; FileName: STR12;
  ListName: STR08);
    (* Laden *)
PROCEDURE KillDisk (FileName: STR12);
    (* Loescht bel. File von akt. Lw *)
```

Hinweise:

Sämtliche Routinen werden nur ausgeführt, wenn die Variable NotExist = FALSE ist! Die Liste der Variablen, die von der jeweiligen Routine verwendet werden, ist im Listing angeführt!

PROCEDURE SetFirstNode
Setzt Current als Zeiger auf das erste Listenelement, CurrentNo = FirstNo;

PROCEDURE SetLastNode
Setzt Current als Zeiger auf das letzte Listenelement, CurrentNo = LastNo;

FUNCTION Next: List
Gibt den Zeiger auf das nächste Listenelement (vom aktuellen Knoten aus gesehen) zurück.

Hinweis:

CurrentNo wird nicht neu gesetzt. Soll eine Liste mit diesem Befehl durchlaufen werden, so muß sich der Programmierer selbst um ein Updaten der Variable kümmern;

FUNCTION Prev: List
Gibt den Zeiger auf das vorhergehende Listenelement (vom aktuellen Knoten aus gesehen) zurück.

Hinweis:

CurrentNo wird nicht neu gesetzt. Soll eine Liste mit diesem Befehl durchlaufen werden, so muß sich der Programmierer selbst um ein Updaten der Variable kümmern;

FUNCTION ListSize: LONGINT;
Liefert die Anzahl Bytes als Ergebnis, die zur Speicherung der gerade aktuellen Liste gebraucht werden.

FUNCTION DataSize: LONGINT;
Liefert die Anzahl Bytes als Ergebnis, die effektiv zur Speicherung der Benutzerdaten gebraucht werden.

FUNCTION ListName: STR08;
Liefert den Listennamen der aktuellen Liste zurück.

PROCEDURE SetActiveList (Name: STR08);
Setzt die Liste mit dem Namen, der in der Variable Name übergeben wird, als aktive Liste. Die Parameter (sämtliche listenspezifische Variable) der vorher aktuellen Liste werden am Liststack abgelegt sowie die der neuen aktuellen Liste geladen.

PROCEDURE ML (VAR Data; Name: STR08);
Legt eine neue, einelementige Liste unter dem Namen, der in der Variable Name angegeben wird, an und speichert die Parameter am Liststack (hier auf maximal 100 Listen gleichzeitig dimensioniert).

PROCEDURE KillMem;
Löscht die momentan gesetzte Liste auf dem Heap, sämtliche Parameter werden aus dem Liststack gelöscht, die Variable TopOfList um 1 erniedrigt.

Hinweis:

nach einem Aufruf von KillMem ist natürlich keine Liste aktiv.

PROCEDURE LPut (VAR Data);
Fügt die Variable, die mit Data übergeben wird, als Knoten an das Listenende an (LastPut). LastNo wird inkrementiert.

PROCEDURE FPut (VAR Data);
Fügt die Variable, die mit Data übergeben wird, als Knoten an den Listenanfang an (FirstPut). LastNo und CurrentNo werden inkrementiert.

PROCEDURE InsCur (VAR Data);
Fügt die Variable, die mit Data übergeben wird, als Knoten an der gerade aktuellen Stelle ein, LastNo wird inkrementiert.

Hinweis:

Diese Procedure arbeitet nicht, wenn der gerade aktive Knoten das erste oder letzte Element in der Liste ist.

PROCEDURE DelCur;
Löscht das Element, auf das Current gerade zeigt (=das aktuelle Element), aus der aktiven Liste. handelt es sich um eine einelementige Liste (LastNo=1), so wird automatisch ein KillMem ausgeführt.

PROCEDURE GetNext (VAR Data);
Liefert über die Variable Data den Datenteil des nächsten Listenelementes, Current und CurrentNo werden neu gesetzt.

Hinweis:

wird eine Liste linear mit GetNext durchlaufen, so ist darauf zu achten, daß GetNext nicht auf das letzte Listenelement angewandt wird.

PROCEDURE GetPrev (VAR Data);
Liefert über die Variable Data den Datenteil des vorhergehenden Listenelementes, Current und CurrentNo werden neu gesetzt.

Hinweis:

wird eine Liste linear mit GetPrev durchlaufen, so ist darauf zu achten, daß GetPrev nicht auf das erste Listenelement angewandt wird.

PROCEDURE GetData (VAR Data);
Gibt über Data den Datenteil des gerade aktuellen Knotens zurück.

PROCEDURE PutData (VAR Data);
Überschreibt den Datenteil des gerade aktuellen Knotens mit dem Inhalt von Data.

PROCEDURE SeekNo (No: WORD);
Sucht auf dem kürzesten Weg linear nach dem Listenelement mit der Nummer No, setzt Current und CurrentNo neu.

PROCEDURE SaveList (VAR Buf; FileName: STR12);
Speichert die gerade aktive Liste unter dem mit FileName angegebenen Namen auf dem gerade gesetzten Laufwerk. Die Speicherung erfolgt blockweise, ein Block entspricht einem Listenknoten. Die Statusvariable IORes wird gesetzt.

PROCEDURE LoadList (VAR Buf; FileName: STR12; ListName: STR08);
Ladet das File mit dem in der Variablen FileName angegebenen Namen als Liste mit dem in ListName angegebenen Namen. Existiert im Liststack eine Liste gleichen Namens, so wird keine Liste geladen. Die Statusvariable IORes wird gesetzt.

Hinweis:

Vor der Verwendung von LoadList muß unbedingt ObjectSize initialisiert werden.

PROCEDURE KillDisk (FileName: STR12);
Löscht das File mit dem in der Variable FileName angegebenen Namen vom gerade aktuellen Laufwerk. Die Statusvariable IORes wird gesetzt.

4.3) Interne Details der UNIT List_Man:

4.3.1) Interne Variablendefinitionen:

```
CONST MaxList = 101;
TYPE STR08 = STRING [8];

     ListenName = RECORD
       LName: STR08;                (* Speicherung des Listennamens *)
       FPtr, LPtr, Cur      : List;  (* Speicherung der Listenpointer*)
       FNo, LNo, CurNo, ObjS: WORD;  (* weitere Listendaten          *)
     END; (* RECORD ListenName *)

VAR FirstPtr, LastPtr: List;
    ListField        : ARRAY [0..MaxList] OF ListenName;
```

FirstPtr: List;
zeigt auf den ersten Knoten der aktiven Liste.

LastPtr: List;
zeigt auf den letzten Knoten der aktiven Liste.

ListField: ARRAY [0..MaxList] OF ListenName;
Stellt den Liststack dar, d.h. in diesem Array, bestehend aus einzelnen Records (ListenName), werden sämtliche Informationen der sich gerade im Heap befindlichen Listen gespeichert.

Die Konstante MaxList:
Mit MaxList wird die Anzahl der Listen festgelegt, die sich maximal gleichzeitig im Heap befinden dürfen (bei MaxList = 101 - 100 Listen).

4.3.2) Speicherung eines Knotens:

Um beliebige Datenstrukturen innerhalb eines Knotens zu ermöglichen, wurde folgender Lösungsweg gegangen:

Ein Knoten besteht aus drei Pointern

```
TYPE List = ^ListElement;
     ListElement = RECORD
       Next, Prev: List;
       DataPtr   : POINTER;
     END; (* RECORD ListElement *)
```

Next, Prev sind Pointer auf den jeweils nächsten bzw. vorhergenden Knoten in der Liste.

DataPtr ist ein Zeiger auf den eigentlichen Datensatz, d.h. die Speicherung der Benutzerdaten erfolgt nicht innerhalb eines Listenknotens, sondern extern im Heap. Innerhalb eines Knotens wird demnach nur ein Zeiger auf den zugehörigen im Heap gespeicherten Datensatz erzeugt.

Die Reservierung des Speicherplatzes für einen Benutzerdatensatz muß aus Gründen der Flexibilität der UNIT via GetMem erfolgen (bei NEW muß die Datensatzgröße für alle Zeiten von vornherein feststehen, jeder Benutzer möchte jedoch andere Daten in einer Liste speichern (Textdatei, Meßtabellen...)).

Daher muß über die in Turbo Pascal vordefinierte Funktion SizeOf (oder anderweitig) die Größe eines Records ermittelt und in der vordefinierten Variable ObjectSize abgelegt werden (GetMem reserviert einen Speicherblock, der Größe ObjectSize).

Die Startadresse des mittels GetMem belegten Speicherblocks wird dann in dem aktuellen ListNode als Pointer (in DataPtr) gespeichert. So ist es möglich auch über eigene, (noch) nicht in der UNIT List_Man definierte Routinen auf diesen Datenbereich zuzugreifen.

Die Speicherung der Daten aus der angegebenen Variablen in den reservierten Speicherbereich wird dann über den MOVE-Befehl abgewickelt.

Die Löschung eines Datensatzes erfolgt dann umgekehrt analog zur Reservierung.

4.3.3) Der Liststack:

Hierbei handelt es sich nicht, wie die Überschrift vermuten läßt, um einen Stack im herkömlichen Sinne (LIFO), sondern um ein Recordarray, das pro Arrayelement (= ein RECORD) sämtliche listenspezifischen Daten enthält:

```
TYPE STR08 = STRING [8];

ListenName = RECORD
  LName: STR08;                (* Speicherung des Listennamens *)
  FPtr, LPtr, Cur      : List;  (* Speicherung der Listenpointer*)
  FNo, LNo, CurNo, ObjS: WORD;  (* weitere Listendaten          *)
END; (* RECORD ListenName *)
```

Die einzelnen Arrayelemente wurden als RECORD folgendermaßen definiert:

Die Anzahl der Listen, die gleichzeitig im Speicher gehalten werden können, kann über die Konstante MaxDaten im INTERFACE-Teil der UNIT festgelegt werden. Zu beachten ist dabei nur, daß die Konstante um 1 größer sein muß als dieègewünschte Anzahl Listen (siehe KillMem im Listing).

Der Liststack wird von den folgenden Routinen verwaltet:

SetActiveList:
speichert die Parameter der aktuellen Liste und setzt die in der Variable Name angegebene Liste als aktuelle Liste (das Array wird auf den Listennamen hin abgesucht, bis TopOfList).

ML (MakeList):
wird MakeList aufgerufen, so werden ebenfalls die Parameter der gerade aktuellen Liste gespeichert, die Variable TopOfList inkrementiert und sämtliche Parameter der neuen Liste in das zugehörige, über TopOfList angegebene, Record eingetragen.

KillMem:
löscht die aktive Liste; alle Arrayelemente von ActiveListNo bis TopOfList werden jeweils in das logisch "vor ihnen" liegende kopiert.

4.4.2) Testprogramm (dt2.pas):

4.4) Beispiele:

4.4.1) Ersatzteilverwaltung (letzte Version!):

```
PROGRAM Verwalte5;
USES CRT;

TYPE Ersatzteil = RECORD
       Bezeichnung: STRING[100];          {100  Bytes }
       Artikelnr  : LONGINT;              {  4  Bytes }
       Anzahl     : WORD;                 {  2  Bytes }
       Preis      : WORD;                 {  2  Bytes }
     END; (* RECORD Ersatzteil *)         { Gesamt: 108  Bytes/RECORD }
VAR User: Ersatzteil;
    Name: STR08;
    Ende: CHAR;

PROCEDURE Read_Record (VAR u: Ersatzteil);
BEGIN
  WriteLn ('Einlesen des Ersatzteil Datensatz mit Nr: ',(CurrentNo+1));
  Write ('Ersatzteil     : '); ReadLn (u.Bezeichnung);
  Write ('Artikelnummer  : '); ReadLn (u.Artikelnr);
  Write ('Anzahl         : '); ReadLn (u.Anzahl);
  Write ('Preis          : '); ReadLn (u.Preis);
END; (* PROC Read_Record *)

PROCEDURE Write_Record (u: Ersatzteil);
BEGIN
  WriteLn ('Zugriff auf den Ersatzteil Datensatz mit Nr: ',CurrentNo);
  WriteLn ('Ersatzteil     : ', u.Bezeichnung);
  WriteLn ('Artikelnummer  : ', u.Artikelnr);
  WriteLn ('Anzahl         : ', u.Anzahl);
  WriteLn ('Preis          : ', u.Preis);
END; (* PROC Write_Record *)

BEGIN
  ...
  < S O U R C E >
  ...
  Read_Record (User);
  ObjectSize:=SizeOf (User);
  ML (User, Name);
  REPEAT
    WriteLn ('Weiter n/Taste : '); Ende:=ReadKey;
    Ende:=UpCase(Ende);
    IF (Ende <> 'N') THEN BEGIN
       Read_Record (User);
       LPut (User);
    END;
  UNTIL (Ende = 'N');
  ...
  < S O U R C E >
  ...
  SetFirstNode;
  GetData (User);
  Write_Record (User);
  REPEAT
    GetNext (User);
    Write_Record (User);
    WriteLn ('Weiter n/Taste : '); Ende:=ReadKey;
    Ende:=UpCase(Ende);
  UNTIL ((Next = NIL) OR (Ende = 'N')); (* bis Listenende od. Ende*)
  ...
  < S O U R C E >
  ...
END. (* PROGRAM Verwalte5 *)
```

4.4.2) Testprogramm (dt2.pas):

```
PROGRAM Dt2;
USES DOS, CRT, List_Man;

TYPE Name = RECORD       (* TestRecord definieren *)
       VN: STRING[20];
       NN: STRING[20];
     END;

VAR H1, H2, H3, H4     : LONGINT;         (* Anzahl freier Bytes am HEAP *)
    Name1, Name2, Name3: STR08;           (* Listennamen                 *)
    s                  : STRING[100];     (* Testvariable *)
    r                  : Name;            (* Testvariable *)
    li                 : LONGINT;         (* Testvariable *)
    Hour, Min, Sec, S100, MaxNodes: WORD; (* Var. f. Zeitmessung, MaxNodevar *)

PROCEDURE Line;                           (* Zieht eine horiz. Linie *)
VAR x: BYTE;
BEGIN
  FOR x:=1 TO 80 DO Write (#196);
END; (* PROC Line *)

PROCEDURE Write_Title;
BEGIN
  ClrScr;
  WriteLn ('List_Manager Demonstrationsprogramm:  (c) e.huber');
  Line;
END; (* PROC Write_Title *)

PROCEDURE GiveStatistics;    (* Gibt einige Zahlen, den Heap betreffend, aus *)
BEGIN
  Line;
  WriteLn ('Statistik:');
  WriteLn ('Gesamt verfuegbarer Platz am Heap       : ',H1:6,' Bytes');
  WriteLn ('Platz belegt durch die gesamte/n Liste/n: ',(H1-H3):6,' Bytes');
  WriteLn ('Noch freier Speicherplatz               : ',H3:6,' Bytes');
  WriteLn ('Verfügbarer Platz nach Löschen der Liste: ',H4:6,' Bytes');
  WriteLn ('                                Weiter mit RETURN');
  Line;
  ReadLn;
END; (* PROC Statistics *)


BEGIN
H1:=MemAvail;
Name1:='Name1'; Name2:='Name2'; Name3:='Name3';
s:='My Name is';
r.VN:='Ernst'; r.NN:='HUBER';

Write_Title;

ObjectSize:=SizeOf (li);  (* ObjectSize init. - Größe des Datenfeldes best. *)
ML (li, Name1);           (* Einelementige Liste anlegen                    *)

MaxNodes:=(MemAvail DIV (13+ObjectSize)); (* Max. mögl. Anz. d. Knoten best.*)

Write ('Gen. Liste mit ',MaxNodes:6,' Nodes (LongInt)    : ');
GetTime (Hour,Min,Sec,S100);
```

```
WriteLn (Hour:2,' h ',Min:2,' m ',Sec:2,' s ',S100:2);

REPEAT                    (* Liste solange erweitern, bis die max. mögl.   *)
  li:=CurrentNo;          (* Anzahl von Knoten erreicht ist                *)
  LPut (li);
UNTIL ((CurrentNo >= MaxNodes) OR (NotExist));

GetTime (Hour,Min,Sec,S100);
Write ('Liste mit ',MaxNodes:6,' Knoten erzeugt          : ');
WriteLn (Hour:2,' h ',Min:2,' m ',Sec:2,' s ',S100:2);
H3:=MemAvail;

Write ('Speichere Liste auf dem gerade aktiven Lw: ');
GetTime (Hour,Min,Sec,S100);
WriteLn (Hour:2,' h ',Min:2,' m ',Sec:2,' s ',S100:2);
SaveList (li,'test.lst');

GetTime (Hour,Min,Sec,S100);
Write ('Fertig mit SaveList/beginne mit KillMem  : ');
WriteLn (Hour:2,' h ',Min:2,' m ',Sec:2,' s ',S100:2);
KillMem;

GetTime (Hour,Min,Sec,S100);
Write ('Fertig mit KillMem/beginne mit LoadList  : ');
WriteLn (Hour:2,' h ',Min:2,' m ',Sec:2,' s ',S100:2);
ObjectSize:=SizeOf(li);
LoadList (li,'test.lst',Name1);

GetTime (Hour,Min,Sec,S100);
Write ('Fertig mit LoadList                      : ');
WriteLn (Hour:2,' h ',Min:2,' m ',Sec:2,' s ',S100:2);
KillMem;
H4:=MemAvail;
GiveStatistics;

H1:=MemAvail;
Write_Title;
WriteLn ('Erzeuge 3 Listen bestehend aus jeweils 10 Elementen:');
WriteLn (' LONGINT           STRING           RECORD');
WriteLn ('                                    Vorname        Nachname');
li:=0;
ObjectSize:=SizeOf(li);     (* Generiere Liste 1 mit LONGINT-Datenfeld   *)
ML (li,Name1);
REPEAT
  li:=CurrentNo;
  LPut (li);
UNTIL (CurrentNo >= 10);

ObjectSize:=SizeOf(s);      (* Generiere Liste 2 mit String100-Datenfeld *)
ML (s,Name2);
REPEAT
  Lput (s);
UNTIL (CurrentNo >= 10);

ObjectSize:=SizeOf(r);      (* Generiere Liste 3 mit RECORD-Datenfeld    *)
ML (r,Name3);
REPEAT
  LPut (r);
UNTIL (CurrentNo >= 10);

H3:=MemAvail;
```

```
SetActiveList (Name1);                  (* Liste 1 aktivieren          *)
SetFirstNode;                           (* zum Listenangfang springen *)

GetData (li);                           (* 1. Datensatz holen          *)
GotoXY (5,6+CurrentNo); WriteLn (li);   (* und am Bildschirm ausgeben *)
REPEAT
  GetNext (li);                         (* nächsten Datensatz holen    *)
  GotoXY (5,6+CurrentNo); WriteLn (li); (* am Bildschirm ausgeben,     *)
UNTIL (Next=NIL);                       (* solange bis Listenende      *)

s:='';
SetActiveList (Name2);                  (* wie bei Liste 1             *)
SetFirstNode;
GetData (s);
GotoXY (19,6+CurrentNo); WriteLn (s);
REPEAT
  GetNext (s);
  GotoXY (19,6+CurrentNo); WriteLn (s);
UNTIL (Next=NIL);

r.VN:=''; r.NN:='';                     (* wie bei Liste 1             *)
SetActiveList (Name3);
SetFirstNode;
GetData (r);
GotoXY (39,6+CurrentNo); WriteLn (r.VN,'          ',r.NN);
REPEAT
  GetNext (r);
  GotoXY (39,6+CurrentNo); WriteLn (r.VN,'          ',r.NN);
UNTIL (Next=NIL);

SetActiveList (Name1); KillMem;   (* Liste1 aktivieren und löschen *)
SetActiveList (Name2); KillMem;   (* Liste2 aktivieren und löschen *)
SetActiveList (Name3); KillMem;   (* Liste3 aktivieren und löschen *)
H4:=MemAvail;
GiveStatistics;

END. (* PROGRAM DT2 *)
```

UNIT GRAPH_IO

TGM_96: GRAPH.TXT, TGM_97:GRAPH.ARC

Ernst Huber, N86d

Die Unit GRAPH_IO stellt eine Anzahl von Bildschirm-Ein/Ausgabeoperationen im Graphikmodus von TURBO PASCAL 4.0 zur Verfügung.

1) Ausgabe-(Write) Routinen

Die UNIT GRAPH_IO stellt 6 Ausgabe- (Write) Routinen zur Ausgabe von verschiedenen Variablentypen zur Verfügung (STRING, INTEGER, LONGINT, WORD, REAL (Fixkomma), REAL (Floating-Point)). Sämtliche Schriftarten, die TURBO PASCAL 4.0 zur Verfügung stellt, werden unterstützt, jedoch soll gleich vorweggenommen werden, daß sich Probleme mit den Unterlängen bei den Vektorschriftarten in verschiedenen Größen ergeben. Am besten geeignet ist der sogen. 'DefaultFont', ein (8x8) Pixel Font, auf den man bei normaler Maskenprogrammierung zurückgreifen sollte, da dieser am besten lesbar ist (hier ergeben sich keine Schwierigkeiten mit den Unterlängen (die UNIT wurde hauptsächlich für diesen Font geschrieben), ebenso bei Verwendung des 'SmallFont'). Ein Ausgabestring (vor der Ausgabe wird jede Variable in einen String umgewandelt und dieser ausgegeben) kann mit einer bestimmten Farbe unterlegt werden (dies sollte im Sinne einer übersichtlichen Bildschirmgestaltung für den Endbenutzer geschehen).

PROCEDURE Wr_String (X, Y, BarCol, Digits, V);

Eingabeparameter:

X, Y: WORD : gibt die Bildschirmposition an, ab der die Textausgabe erfolgen soll;

BarCol: WORD: anhand von BarCol wird die Farbe festgelegt, mit der die Ausgabe unterlegt werden soll;

Digits: WORD: gibt die Anzahl der Stellen an, die ausge- geben werden sollen;

V: STRING : Stringvariable, die ausgegeben wird;

Hinweis:

die Balkenlänge ist ungefähr 1 Digit länger als in der Variable Digits angegeben (vor Textanfang 1/2 Digit, nach Textende 1/2 Digit).

PROCEDURE Wr_Int (X, Y, BarCol, V)

Eingabeparameter:

X, Y: WORD : gibt die Bildschirmposition an, ab der die Textausgabe erfolgen soll;

BarCol: WORD: anhand von BarCol wird die Farbe festgelegt, mit der die Ausgabe unterlegt werden soll;

V: INTEGER : Integervariable, die ausgegeben wird;

Hinweis:

Die Länge des Ausgabefeldes bei Integervariablen beträgt immer 6 + 1 Stellen (6 Stellen für die Variable, 1 Stelle wird aufgeteilt in: vor Textanfang 1/2 Digit, nach Textende 1/2 Digit).

PROCEDURE Wr_Long (X, Y, BarCol, V)

Eingabeparameter:

X, Y: WORD : gibt die Bildschirmposition an, ab der die Textausgabe erfolgen soll;

BarCol: WORD: anhand von BarCol wird die Farbe festgelegt, mit der die Ausgabe unterlegt werden soll;

V: LONGINT : Integervariable, die ausgegeben wird;

Hinweis:

Die Länge des Ausgabefeldes bei LongIntvariablen beträgt immer11 + 1 Stellen (11 Stellen für die Variable, 1 Stelle wird aufgeteilt in: vor Textanfang 1/2 Digit, nach Textende 1/2 Digit).

PROCEDURE Wr_Word (X, Y, BarCol, V)

Eingabeparameter:

X, Y: WORD : gibt die Bildschirmposition an, ab der die Textausgabe erfolgen soll;

BarCol: WORD: anhand von BarCol wird die Farbe festgelegt, mit der die Ausgabe unterlegt werden soll;

V: WORD : Wordvariable, die ausgegeben wird;

Hinweis:

Die Länge des Ausgabefeldes bei Wordvariablen beträgt immer5 + 1 Stellen (5 Stellen für die Variable, 1 Stelle wird aufgeteilt in: vor Textanfang 1/2 Digit, nach Textende 1/2 Digit).

PROCEDURE Wr_Floati (X, Y, BarCol, Digits, V)

Eingabeparameter:

X, Y: WORD : gibt die Bildschirmposition an, ab der die Textausgabe erfolgen soll;

BarCol: WORD: anhand von BarCol wird die Farbe festgelegt, mit der die Ausgabe unterlegt werden soll;

Digits: WORD: gibt die Anzahl der Stellen an, die ausgegeben werden sollen (gesamt); Minimalwert für Digits: 3/4 bei (+ /-) Werten;

V: REAL : Realvariable, die im Floatingpointformat ausgegeben wird;

Hinweis:

die Balkenlänge ist ungefähr 1 Digit länger als in der VariableDigits angegeben (vor Textanfang 1/2 Digit, nach Textende 1/2 Digit).

PROCEDURE Wr_Fix (X, Y, BarCol, Digits, V)

Eingabeparameter:

X, Y: WORD : gibt die Bildschirmposition an, ab der die Textausgabe erfolgen soll;

BarCol: WORD: anhand von BarCol wird die Farbe festgelegt, mit der die Ausgabe unterlegt werden soll;

Digits: WORD: gibt die Anzahl der Stellen an, die ausgegeben werden sollen (gesamt);

V: REAL : Realvariable, die in Fixkommadarstellung ausgegeben wird; Bedingung: V muß im Bereich von 1E-11 bis 1E11 oder -1E + 11 bis -1E-11 liegen. Ist dies nicht der Fall, wird automatisch Wr_Floati ausgeführt;

Hinweis:

die Balkenlänge ist ungefähr 1 Digit länger als in der VariableDigits angegeben (vor Textanfang 1/2 Digit, nach Textende 1/2 Digit).

2) Eingabe-(Read) Routinen:

Ebenso wie bei den Ausgaberoutinen werden für die oben angeführten 6 Variablentypen Eingabe- (Read) Routinen zur Verfügung gestellt. Sämtliche Read-Routinen verlangen eine Variable (Status), in der die jeweils aufgerufene Prozedur einen Returncode ablegt, der dann weiter ausgewertet werden kann. Dieser Returncode (Integer) gibt an, ob der Benutzer eine gültige Eingabe, eine Spezialtaste (Crs, Esc etc.) oder eine (bei numerischen Anwendungen) Zahl außerhalb des zulässigen Wertebereiches eingegeben hat. Zur leichteren Auswertung dieser Variablen wurden bereits einige Werte vordefiniert, die im Punkt 3) näher behandelt werden. Wie bereits angedeutet, bieten sämtliche Read-Routinen, die numerische Werte einlesen, die Möglichkeit, einen Wertebereich festzulegen, innerhalb dessen die einzulesende Variable liegen muß (sonst keine gültige Eingabe, der alte Variablenwert bleibt erhalten). Auch hierfür wurden bereits einige Konstanten vordefiniert, näheres siehe Punkt 3).

Folgende Tasten werden von den Read-Routinen aktzeptiert und folgendermaßen interpretiert:

< ESC > Die Eingabe wird abgebrochen, der alte Variablenwert bleibt erhalten, der Returncode = der Konst. ESC;

< Crs-Down > Die Eingabe wird abgebrochen, der alte Variablenwert bleibt erhalten, der Returncode = der Konst. Crs_Down;

< Crs-Up > Die Eingabe wird abgebrochen, der alte Variablenwert bleibt erhalten, der Returncode = der Konst. Crs_Up;

< Crs-Left > Die Eingabe wird abgebrochen, der alte Variablenwert bleibt erhalten, der Returncode = der Konst. Crs_Left;

< Crs-Right > Die Eingabe wird abgebrochen, der alte Variablenwert bleibt erhalten, der Returncode = der Konst. Crs_Right;

< BackStep > Das zuletzt eingegebene Zeichen wird gelöscht, die Routine wartet auf weiteren Tastendruck;

< Return > Eingabe wird abgeschlossen, eingegebene Variable auf Gültigkeit überprüft und bei Gültigkeit der angegebenen Variablen zugewiesen (der Returncode = Konst. Return). Ist die eingegebene Variable ungültig (bei numerischer Eingabe), dann wird der alte Variablenwert beibehalten. Der Returncode = To_Less/To_Big;

PROCEDURE Rd_String (X, Y, BARCol, Digits, CharSet, Status, V);

Eingabeparameter:

X, Y: Start des Balkens, der als Hintergrund ausgegeben wird;

BarCol : Hintergrundfarbe des Balkens für Stringeingabe;

Digits : gibt die Länge der einzulesenden Variablen an;

CharSet: gibt die Menge der gültigen Zeichen an, alle anderen Zeichen werden ignoriert;

Ausgabeparameter:

Status : gibt einen Returncode ob eine gültige Eingabe oder eine 'Sondertaste' gedrückt wurde;

V: Stringvariable, die eingelesen werden soll;

Hinweis:

die Balkenlänge ist ungefähr 1 Digit länger als in der Variablen Digits angegeben (vor Textanfang 1/2 Digit, nach Textende 1/2 Digit).

PROCEDURE Rd_Int (X, Y, BARCol, MinV, MaxV, Status, V);

Eingabeparameter:

X, Y: Start des Balkens, der als Hintergrund ausgegeben wird;

BarCol : Hintergrundfarbe des Balkens für Integereingabe;

MinV: min. zulässiger Wert der Variablen, die eingegeben werden soll;

MaxV: max. zulässiger Wert der Variablen die eingegeben werden soll;

Ausgabeparameter:

Status : gibt einen Returncode, ob eine gültige Eingabe oder eine 'Sondertaste' gedrückt wurde;

V: Integervariable, die eingelesen werden soll;

Hinweis:

Die Länge des Eingabefeldes bei Integervariablen beträgt immer6 + 1 Stellen (6 Stellen für die Variable, 1 Stelle wird aufgeteilt in: vor Textanfang 1/2 Digit, nach Textende 1/2 Digit).

PROCEDURE Rd_Long (X, Y, BARCol, MinV, MaxV, Status, V);

Eingabeparameter:

X, Y: Start des Balkens, der als Hintergrund ausgegeben wird;

BarCol : Hintergrundfarbe des Balkens für LongInteingabe;

MinV: min. zulässiger Wert der Variablen, die eingegeben werden soll;

MaxV: max. zulässiger Wert der Variablen, die eingegeben werden soll;

Ausgabeparameter:

Status : gibt einen Returncode, ob eine gültige Eingabe oder eine 'Sondertaste' gedrückt wurde;

V: LongIntvariable, die eingelesen werden soll;

Hinweis:

Die Länge des Eingabefeldes bei LongIntvariablen beträgt immer 11 + 1 Stellen (11 Stellen für die Variable, 1 Stelle wird aufgeteilt in: vor Textanfang 1/2 Digit, nach Textende 1/2 Digit).

PROCEDURE Rd_Word (X, Y, BARCol, MinV, MaxV, Status, V);

Eingabeparameter:

X, Y: Start des Balkens, der als Hintergrund ausgegeben wird;

BarCol : Hintergrundfarbe des Balkens für Wordeingabe;

MinV: min. zulässiger Wert der Variablen, die eingegeben werden soll;

MaxV: max. zulässiger Wert der Variablen, die eingegeben werden soll;

Ausgabeparameter:

Status : gibt einen Returncode, ob eine gültige Eingabe oder eine 'Sondertaste' gedrückt wurde;

V: Variable des Typs WORD, die eingelesen werden soll;

Hinweis:

Die Länge des Eingabefeldes bei Wordvariablen beträgt immer5 + 1 Stellen (5 Stellen für die Variable, 1 Stelle wird aufgeteilt in: vor Textanfang 1/2 Digit, nach Textende 1/2 Digit).

PROCEDURE Rd_Floati (X, Y, BARCol, Digits, MinV, MaxV, Status, V);

Eingabeparameter:

X, Y: Start des Balkens, der als Hintergrund ausgegeben wird;

BarCol : Hintergrundfarbe des Balkens für Realeingabe;

Digits : gibt die Länge der einzulesenden Variable an

MinV: min. zulässiger Wert der Variablen, die eingegeben werden soll;

MaxV: max. zulässiger Wert der Variablen, die eingegeben werden soll;

Ausgabeparameter:

Status : gibt einen Returncode, ob eine gültige Eingabe oder eine 'Sondertaste' gedrückt wurde;

V: Variable des Typs REAL, die eingelesen werden soll;

Hinweis:

die Balkenlänge ist ungefähr 1 Digit länger als in der Variablen Digits angegeben (vor Textanfang 1/2 Digit, nach Textende 1/2 Digit).

PROCEDURE Rd_Fix (X, Y, BARCol, Digits, MinV, MaxV, Status, V);

Eingabeparameter:

X, Y: Start des Balkens, der als Hintergrund ausgegeben wird;

BarCol : Hintergrundfarbe des Balkens für Realeingabe;

Digits : gibt die Länge der einzulesenden Variable an

MinV: min. zulässiger Wert der Variablen, die eingegeben werden soll;

MaxV: max. zulässiger Wert der Variablen, die eingegeben werden soll;

Ausgabeparameter:

Status : gibt einen Returncode, ob eine gültige Eingabe oder eine 'Sondertaste' gedrückt wurde;

V: Variable des Typs REAL, die eingelesen werden soll;

Hinweis:

die Balkenlänge ist ungefähr 1 Digit länger als in der VariableDigits angegeben (vor Textanfang 1/2 Digit, nach Textende 1/2 Digit).

3) Typ- und Konstantendefinitionen:
Zur komfortablen Auswertung der Statusvariable sowie zur Bereichsfestlegung bei den Read-Routinen wurden einige Konstanten und Typen bereits vordekla- riert.

3.1) Typendefinitionen:
Whole_Set = SET OF #0..#255;

Als Whole_Set wurde eine Menge definiert, die den gesamten ASCII-Zeichenvorrat umfaßt. Bei Read-String können dann Teilmengen übergeben werden.

Liegt ein Zeichen, das eingegeben wurde, nicht in der ang. Menge, so wird es ignoriert.

3.2) Konstantendefinitionen:
Der Returncode der Statusvariablen kann folgende Werte annehmen:

To_Less : INTEGER = 1;
nur bei num. Eingaben, wenn der eingegebene Wert kleiner als der mit MinV angegebene Wert ist;
To_Big: INTEGER = 2;
nur bei num. Eingaben, wenn der eingegebene Wertgröße als der mit MaxV angegebene Wert ist;
Esc: INTEGER = 3;
keine neue Eingabe, Variable bleibt erhalten, ESCAPE wurde gedrückt;
Crs_Down : INTEGER = 4;
keine neue Eingabe, Variable bleibt erhalten, die CURSOR-DOWN Taste wurde gedrückt;
Crs_Up: INTEGER = 5;
keine neue Eingabe, Variable bleibt erhalten, die CURSOR-UP Taste wurde gedrückt;
Crs_Left : INTEGER = 6;
keine neue Eingabe, Variable bleibt erhalten, die CURSOR-LEFT Taste wurde gedrückt;
Crs_Right: INTEGER = 7;
keine neue Eingabe, Variable bleibt erhalten, die CURSOR-RIGHT Taste wurde gedrückt;
Back_Step: INTEGER = 8;
dient nur für interne Zwecke;
Return: INTEGER = 9;
gültige Eingabe, die mit RETURN abgeschlossen wurde;
Number: SET OF '+'..'E' = ['+','-','.','0'..'9','E'];
gibt die Menge aller möglichen Zeichen bei Zahleneingabe an;
Character: SET OF '!'..'~' = ['!'..'~'];
gibt die Menge aller möglichen Zeichen bei Stringeingabe an;

Folgende Konstanten wurden vordefiniert, falls der Programmierer den vollen Zahlenbereich bei einer typisierten Eingabe ausnutzen möchte (diese Konstanten können direkt für MinV/MaxV eingesetzt werden);

```
Min_W    : WORD = 0;              : min. Wert von WORD-Variablen;
Max_W    : WORD = 65535;          : max. Wert von WORD-Variablen;
Min_I    : INTEGER = -32768;      : min. Wert von INTEGER-Variablen;
Max_I    : INTEGER = 32767;       : max. Wert von INTEGER-Variablen;
Min_LI   : LONGINT = -2147483647;: min. Wert von LONGINT-Variablen;
Max_LI   : LONGINT = 2147483647; : max. Wert von LONGINT-Variablen;
Min_R    : REAL = -1.5E+38;       : min. Wert von REAL-Variablen; (für Rd_Fix)
Max_R    : REAL = 1.5E+38;        : max. Wert von REAL-Variablen;   &. Rd_Floati)
```

4) Der INTERFACE-Teil der UNITS GRAPH_IO:

```
(*-------------------------------------------------------------------------*)
(********* UNIT Graph_IO: Ein/Ausgabe von Variablen im Graphikmodus ********)
(********* (c) Juli 1988 by eh                          Ernst HUBER ********)
(*-------------------------------------------------------------------------*)
UNIT Graph_IO;
INTERFACE
USES CRT, GRAPH;

(*-------------------------------------------------------------------------*)
(******************** Typendefinitionen fuer Unit Graph_IO *****************)
(*-------------------------------------------------------------------------*)
TYPE Whole_Set = SET OF  #0..#255;

(*-------------------------------------------------------------------------*)
(***************** Konstantendefinitionen fuer Unit Graph_IO ***************)
(*-------------------------------------------------------------------------*)
```

```
CONST To_Less  : INTEGER = 1;
      To_Big   : INTEGER = 2;
      Esc      : INTEGER = 3;
      Crs_Down : INTEGER = 4;
      Crs_Up   : INTEGER = 5;
      Crs_Left : INTEGER = 6;
      Crs_Right: INTEGER = 7;
      Back_Step: INTEGER = 8;
      Return   : INTEGER = 9;

      Number   : SET OF '+'..'E'=['+','-','.','0'..'9','E'];
      Character: SET OF '!'..'±'=['!'..'±'];

      Min_W    : WORD = 0;
      Max_W    : WORD = 65535;
      Min_I    : INTEGER = -32768;
      Max_I    : INTEGER = 32767;
      Min_LI   : LONGINT = -2147483647;
      Max_LI   : LONGINT = 2147483647;
      Min_R    : REAL = -1.5E+38;
      Max_R    : REAL = 1.5E+38;

(*-------------------------------------------------------------------------*)
(********** oeffentliche Proceduredefinitionen fuer Unit Graph_IO **********)
(*-------------------------------------------------------------------------*)
PROCEDURE Wr_String (X, Y, BARCol, Digits: WORD; V: STRING);
PROCEDURE Wr_Int (X, Y, BARCol: WORD; V: INTEGER);
PROCEDURE Wr_Long (X, Y, BARCol: WORD; V: LONGINT);
PROCEDURE Wr_Word (X, Y, BARCol, V: WORD);
PROCEDURE Wr_Floati (X, Y, BARCol, Digits: WORD; V: REAL);
PROCEDURE Wr_Fix (X, Y, BARCol, Digits: WORD; V: REAL);

PROCEDURE Rd_String (X, Y, BARCol, Digits: Word; CharSet: Whole_Set;
                     VAR Status: BYTE; VAR V: STRING);
PROCEDURE Rd_Int (X, Y, BARCol: WORD; MinV, MaxV: INTEGER;
                  VAR Status: BYTE; VAR V: INTEGER);
PROCEDURE Rd_Long (X, Y, BARCol: WORD; MinV, MaxV: LONGINT;
                   VAR Status: BYTE; VAR V: LONGINT);
PROCEDURE Rd_Word (X, Y, BARCol, MinV, MaxV: Word;
                   VAR Status: BYTE; VAR V: Word);
PROCEDURE Rd_Floati (X, Y, BARCol, Digits: WORD; MinV, MaxV: REAL;
                     VAR Status: BYTE; VAR V: REAL);
PROCEDURE Rd_Fix (X, Y, BARCol, Digits: WORD; MinV, MaxV: REAL;
                  VAR Status: BYTE; VAR V: REAL);

IMPLEMENTATION
(*-------------------------------------------------------------------------*)
(****************************** < S O U R C E > ***************************)
(*-------------------------------------------------------------------------*)
END. (* UNIT Graph_IO *)
```

CLUBTEIL

Sehr geehrtes Clubmitglied !

Als Beilage dieser PC - NEWS finden Sie einen Zahlschein.

Diesen verwenden Sie bitte zur Begleichung des Mitgliedsbeitrages für das Jahr 1989.

Hierbei gilt, daß alle Mitgliedsbeiträge des Jahres 1988 ab der Mitgliedsnummer 1163 (inklusive) schon für 1989 gelten.

Wenn Ihnen die Höhe Ihres Beitrages unklar ist, so nehmen Sie bitte in folgendem Feld Einsicht.

Mitgliedsart	Normal	Schüler	Studenten	Fördernd
Einschreibgeb	300.-	-0-	-0-	-0-
Beitrag/Jahr	300.-	150.-	300.-	1000.-

Schüler und Studenten: Bitte senden Sie eine Schulbesuchs- bzw. Inskriptionsbestätigung an uns !

Bitte beachten Sie auch einen eventuellen Sprung in Ihrem Mitgliedsstatus: Schüler - Student - Normal - Fördernd !

Der Großteil der Arbeit wird von freiwilligen Mitarbeitern

übernommen, bezahlen Sie daher den Mitgliedsbeitrag so rasch als möglich, die Buchhaltung wird dadurch sehr vereinfacht.

Mit freundlichen Grüßen

Ihr P C C - T G M

BANKVERBINDUNG: CA 1200 Wallensteinpl., BLZ.11000 KtNr: 0972-38638/00

In eigener Sache:

Das Setzen einer Clubzeitschrift mittels Desk-Top-Publishing-Programms macht viel (ehrenamtliche) Arbeit. An dieser Stelle wurde schon einmal gebeten, diese Arbeit dadurch zu vermindern, daß die Texte in reinem ASCII-Format und nach gewissen Grundsätzen geschrieben sind (siehe PC-NEWS 2/1988, Seite 3).

Leider mußte auch bei dieser Ausgabe wieder die Erfahrung gemacht werden, daß die Beiträge kaum den Richtlinien entsprechen. Der "Setzer" (Walter Riemer) appelliert daher noch einmal an alle "Schreiber":

Bitte bedenken Sie folgende einfache Rechnung: Für 20 Beiträge kann das Vorformatieren z.B. 20 Stunden dauern (oft auch mehr, weil man nicht immer alle Formatierfehler am Anfang bemerkt, dann wieder aus dem Publisher aussteigen muß, den Texteditor aktivieren, ausbessern, wieder in den Publisher zurück usw. Wenn jeder Autor seinen Beitrag formatiert, ist diese Arbeit in gerechter Weise aufgeteilt, finden Sie nicht ?

TGM-Disks

Nachstehend die Directories der die Hefte 4/1988 und 1/19889 enthaltenden Club-Disketten:

```
TGM_96: PC-News 12 und 13, 4/88 und 1/89 Texte

BASLIST  TXT      791  13.11.88  17.46
BAZAR    TXT      397  13.11.88  23.38
FRAGE    TXT      988  13.11.88  23.36
FWRUNTST TXT     2582  13.11.88  19.00
GRAPH    TXT    19544  13.11.88  23.27
HERCULES TXT     5399  13.11.88  23.28
IC       TXT     3040  13.11.88  17.45
INFO     DFV     1024   8.11.88  18.53
INFO     WD     51712  13.11.88  17.26
KLEIN    TXT     1446  13.11.88  17.48
LC10     TXT     4176  13.11.88  23.28
LISTEN   TXT    45724  13.11.88  23.28
MUMATH   TXT     4623  13.11.88  23.29
MUSIMP   TXT    17915  13.11.88  23.29
PHOENIX  TXT      755  13.11.88  18.13
RAMDISK  TXT     3528  13.11.88  18.14
SYSINFO  TXT     7658  13.11.88  19.10
TAIWAN   TXT     1285  13.11.88  18.15
TELEFON  TXT     2297  13.11.88  23.30
TEX      TXT    16142  13.11.88  18.25
TURBO4   TXT    43794  13.11.88  18.26
VIREN    TXT    17336  13.11.88  18.56
ZGEN     TXT    14700  13.11.88  23.30

TGM_097: PC-News 12 und 13, 4/88 und 1/89 Programme

GRAPH    ARC    71846   8.11.88  20.30
   CGA.BGI          6029
   EGAVGA.BGI       5139
   F_FIALA2.DOC     1553
   GRAPH_IO.DOC    19620
   GRAPH_IO.PAS    23448
   GRIOTEST.EXE    33408
   GRIOTEST.PAS     5899
   HERC.BGI         5933
   LITT.CHR         2126
   SANS.CHR         5452
   TRIP.CHR         7213
HGC      ARC     1538   8.11.88  20.36
   HGC.COM          1591
LC10     ARC    17370   8.11.88  20.33
   STARLC10.DBS    18853
   STARLC10.HLP     7680
   STARNL.DBS       5168
   STARNL10.DBS     7286
LISTEN   ARC    33835   8.11.88  20.35
   DT2.PAS          3824
   DYN_TEST.PAS     4430
   LISTEN.DOC      45649
   LISTENF.DOC      1806
   LIST_MAN.PAS    26622
MUMATH   ARC    38076   8.11.88  20.32
   MUMATH.DOC      81594
MUSIMP   ARC    34661   8.11.88  20.32
   MUSIMPRG.DOC    53837
   PROG1.DOC        6199
```

```
   PROG2.DOC          12411
TELEFON  ARC       4665   8.11.88  20.34
   AUSBES.PRG          1191
   BILD.PRG             630
   DRUCK.PRG           1577
   LOESCHEN.PRG        1705
   MASK.PRG            1098
   MENUE.PRG            780
   NAME.NDX            1024
   TEL.DBF              729
ZGEN     ARC      53706   8.11.88  20.35
   DEMO1.ZEI           4320
   DEMO2.ZEI           4320
   DEMO3.ZEI           4320
   DEMO4.ZEI           4320
   KREIS.ZEI           4320
   KREIS.ZGN             59
   QUADRAT.ZEI         4320
   QUADRAT.ZGN           59
   TEST.ZGN             236
   ZGEN.BAS           10448
   ZGEN.EXE           46064
   ZGEN.TXT           14756
```

Fragen ohne Antworten

Frage: Suche Clipper-Toolbox-Unterlagen bzw. Disketten, Rendl Siegfried, MNum: 1095, BTX 916210678

Frage: Star-Manager-PC: Wenn jemand dieses Programm kennt und benützt, bitte um Kontaktaufnahme mit Dr. Siegfried Pflegerl, Istanbul

Frage: Hat jemand die Dokumentation von Desqview (Quarterdeck)? Antworten erbeten an Paul M. Zulehner, Jagdschloßgasse 16/11, 1130 Wien

Frage: Suche MC 68000 Cross-Assembler, der ähnlich wie ASMZ80 läuft. Antworten erbeten an: Martin Stoll, Ölzeltgasse 10/8, 1030 Wien

Frage: Die Redaktion sucht für eine der folgenden Ausgaben der PC-NEWS Beschreibungen für Geräte, die durch den PC steuerbar sind. Einige Produktbeschreibungen haben wir bereits (Uhr, Taschenrechner, Organizer, Fotoapparat), weitere suchen wir, damit wir eine repräsentative Übersicht über solche Geräte zusammenstellen können.

Frage: Wer kennt ein Programm zum Lesen von ATARI-Disketten am PC? Antworten an Franz FIALA, Siccardsburggasse 4/1/22, 1100 Wien, 784592

.

Mitteilungen

SEIKOSHA SL-80 AI

Herr Smola hat uns das Kopien des Schaltbildes des SEIKOSHA SL-80 AI übermittelt. Interessenten senden wir das Schaltbild gerne zu.

SOFTWARE-BÜCHEREI

Wenn Sie auf der Suche nach speziellen Programmen, die in Österreich etwa noch gar nicht erhältlich sich, nicht weiterkommen, kann Ihnen möglicherweise die SOFTWARE-BÜCHEREI helfen. Sie erhalten die Inhaltsverzeichnisdisketten gratis. Eine Diskette aus diesen Verzeichnissen kostet S 90,-. Anschrift: SOFTWARE-BÜCHEREI, Mollardgasse 46, 1060 Wien

KEINE ERMÄSSIGUNG bei MARKT und TECHNIK

in den vorletzten PC-NEWS (Heft 9/S.35) haben wir einen Preisnachlaß bei Markt und Technik angekündigt, der in den Sommermonaten von der Firma auf Grund einer Reklamation der zuständigen Kammer zurückgenommen wurde. Daher: kein Preisnachlaß bei Büchern von Markt und Technik.

Rabattsätze für PCC-TGM-Mitglieder, von Wolfgang Kollesch

Ich habe Kundenkarten von allen wichtigen Firmen, Mitglieder des PCC-TGM können von mir Ausfolgescheine beziehen (völlig kostenlos) und selbst dort einkaufen, nur gegen Barzahlung. Einige Beispiele für durchschnittliche Rabattsätze, vom Listenpreis:

Elektrogeräte aller Art	43%
Installationsmaterial	47%
Modellspielwaren	44%
Metallartikel	42%
Haushaltsartikel	42%

Weitere Artikel auf Anfrage. Bei Bedarf setzen Sie sich bitte mit Herrn Kollesch (#722) in Verbindung.

Wolfgang Kollesch, Böckhgasse 2-4/18, 1127 Wien

BAZAR

Nachhilfe Nachrichtentechnik, technische Fächer, TGM, Sterz, (0222)/33 63 865, S 100-150/h.

Diskettenschachteln, Kunststoff, hellblau, für ca. 10-15 Disketten 5 1/4" abzugeben, Stückpreis S 20,-, Anfragen an: Helmut Schluderbacher, 62 22 672.

Baby AT-Motherboard S 4000.- 8/10/12 MHz mit Anleitung, ohne RAMs (Herbert König, Tel.35 96 405 und 58801/5380)

TAIWAN

TAIWAN-Importe TGM 96: TAIWAN.TXT
Andreas ZANDOMENEGHI, #589

DIGICOM INC.; 9FL.,109, SEC.4, JEN-AI ROAD TAIPEI TAIWAN R.O.C.
einer der größten Hersteller von Motherboards in Taiwan (siehe auch Beitrag in den vorletzten PCNEWS)

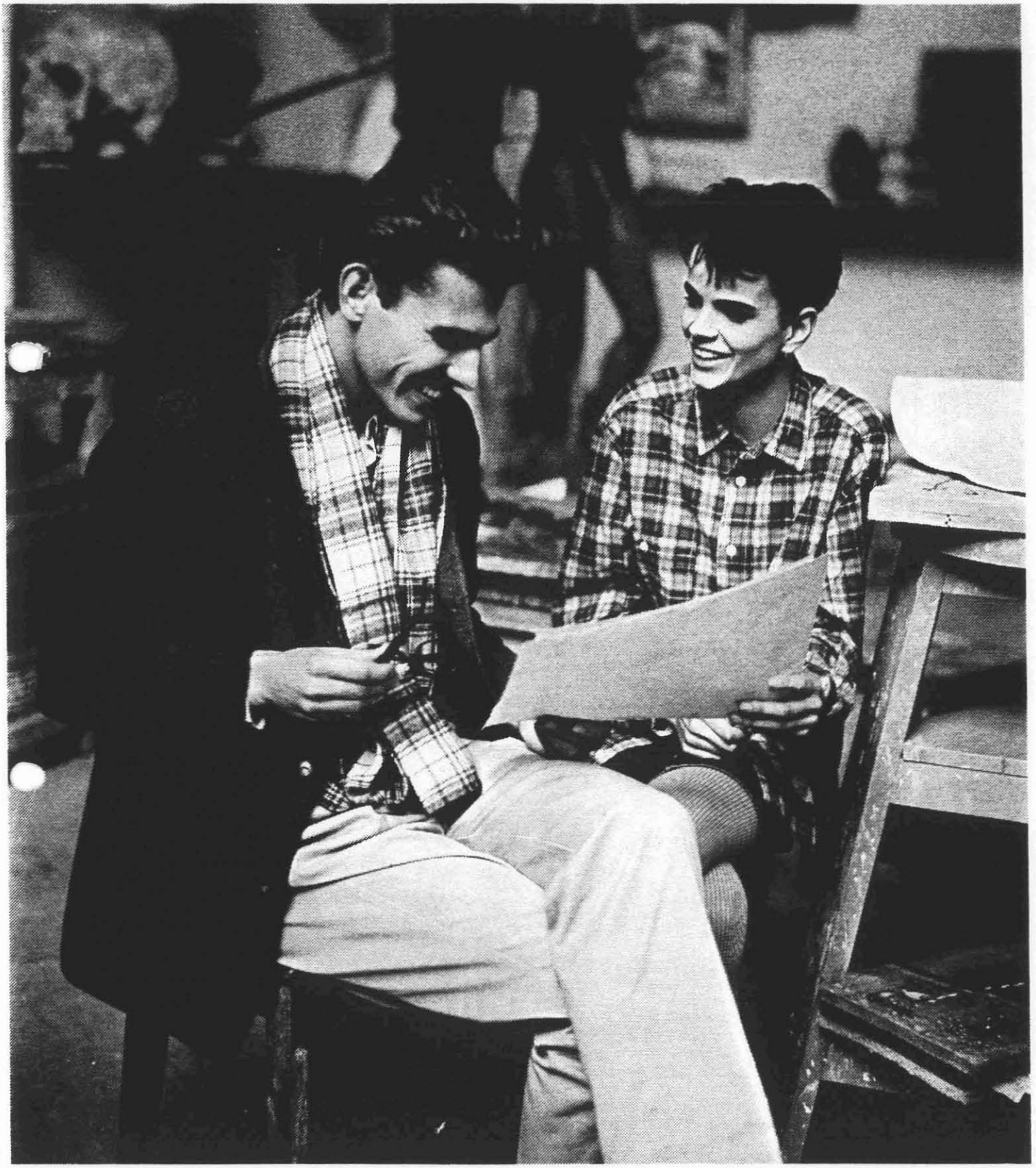
LONGSHINE ELECTRONIC CO.; 6F.,NO.245, SEC.3, ROOSEVELT RD.; TAIPEI TAIWAN R.O.C.
Größter taiwanesischer Hersteller von Controllerboards
FORTUNA ENTERPRISE CO.; 4TH FL.,CHUNGSIN BUILDING; 46/48 FUHSING N.ROAD; TAIPEI TAIWAN R.O.C.
Hat von Platinen bis Joysticks alles im Programm

LADUE LIMITED; 11 F 206 SEC.1, FU HSING S.RD., TAIPEI TAIWAN R.O.C.
Ebenso

QTRONIX CO.; 2RM 11TH FL.,NO.207; TUN HWA N.RD., TAIPEI TAIWAN R.O.C.

Haben u.a. sehr formschöne Mäuse im Programm, sodaß ich dort ein solches "Tier" bestellt habe

CA, die **Bank** *zum* **Erfolg.**



Der Weg zum Erfolg. *Wo ein Wille ist, da findet sich auch ein Weg. Sie wollen mit einem Studium Ihren Weg machen. Und die CA weiß, wie vieles dabei leichter geht. Zum Beispiel mit dem CA-Studentenkonto. CA, die Bank zum Erfolg.*

CREDITANSTALT