

**DYNAMISCHE
DATENSTRUKTUREN
MIT
ZEIGERVARIABLEN**

1. Einführung
2. Lineare Listen
 - 2.1 Vereinbarung einer linearen Liste
 - 2.2 Operationen auf Listen
3. Stapel
4. Weitere dynamische Datenstrukturen

1. Einführung

Problem: Bei einem Schulfest wird ein Wettbewerb im Weit - sprung durchgeführt, an dem möglichst viele Schüler teilnehmen sollen. Die Auswertung und Ermittlung des Siegers soll mit Hilfe eines Computerprogramms erfolgen. Es werden abgespeichert:

- Name
- Vorname
- Sprungweite in cm

(Der Einfachheit halber hat jeder Schüler nur einen gültigen Sprungversuch.)

Das Programm soll folgendes leisten:

- es können prinzipiell beliebig viele Schüler an dem Wettbewerb teilnehmen (1)
- zu jedem Zeitpunkt des Wettbewerbs kann eine aktuelle Rangliste ausgegeben werden (2)
- einzelne Teilnehmer können jederzeit wieder aus der Liste entfernt werden (z.B. bei Disqualifikation) (3)

Die Daten eines Schülers werden in einem Verbund abgespeichert:

```
type datentyp = record
    name      : string[20];
    vorname   : string [15];
    weite     : integer;
end;
```

Da mehrere Schüler teilnehmen sollen, wird eine Datenstruktur benötigt, die viele Elemente des eben vereinbarten Typs aufnehmen kann. Bisher kennen wir den Typ ARRAY

```
TYPE schuelerfeld = array[1..200] of datentyp;
```

Mit dieser Vereinbarung lassen sich die Anforderungen (2) und (3) erfüllen, nicht jedoch die Anforderung (1).

Der Datentyp ARRAY ist ein statischer Datentyp, die Zahl der Elemente wird von vorne herein festgelegt und der entsprechende Speicherplatz im Rechner belegt.

Variablennamen sind symbolische Adressen für die Plätze im Arbeitsspeicher, auf denen die Werte gespeichert sind. Auf die gespeicherten Werte kann man auch mit Zeigern zugreifen.

Zeiger sind Verweise auf ein Datenelement, also Variablen für Adressen, unter denen die Daten abgespeichert sind.

Beispiel:

```
type datensatztyp = record
    name      : string[20];
    vorname   : string [15];
    weite     : integer;
end;

zeigertyp = ^datensatztyp;

var a, b: zeigertyp;
```

Bez.: Die Variable, auf die der Zeiger verweist, heißt Bezugsvariable, ihr Typ Bezugstyp (hier datensatztyp).

Die Variablen a und b nehmen Adressen auf, unter denen Werte vom Datensatztyp gespeichert sind.

Bez.: Das Element, auf das der Zeiger a verweist, ist a^.

```
      +---+
a -----> |a^ |
      +---+
```

Den Umgang mit Zeigervariablen zeigt das folgende Programmbeispiel:

```
program zeigerdemo;

type datensatztyp = record
    name      : string[20];
    vorname   : string [20];
    weite     : integer;
end;

zeigertyp = ^datensatztyp;

var a, b: zeigertyp;

begin new(a);
      new(b);
      write('Name : '); readln(a^.name);
      write('Vorname : '); readln(a^.vorname);
      write('Sprungweite : '); readln(a^.weite);
      b:=nil;
      b:=a;
      writeln('Weite: ',b^.weite);
      dispose(b);
end.
```

Erläuterung: Durch die Prozedur NEW(a) erhält die Zeigervariable a einen Wert, d.h. eine Adresse, unter ein Datenelement gespeichert werden kann. Der Speicherplatz für die Variable a^ wird erst während der Ausführung des Programms (dynamisch) festgelegt.

Die Prozedur DISPOSE(b) gibt den Speicherplatz mit der Adresse des Zeigers wieder frei.

Die einzige Konstante für Zeigervariable ist NIL.

Wertzuweisungen: b := a; bewirkt, daß die beiden Zeiger auf das gleiche Datenelement zeigen. Das Datenelement, auf das b vorher zeigte, ist dann nicht mehr zugänglich.

b^ := a^; bewirkt, daß die Variable b^ den Wert der Variablen a^ erhält.

Übung: Schreiben Sie ein kleines Programm mit einer Bezugsvariablen a^. Untersuchen Sie

- Wertzuweisungen
- READLN
- WRITELN ,

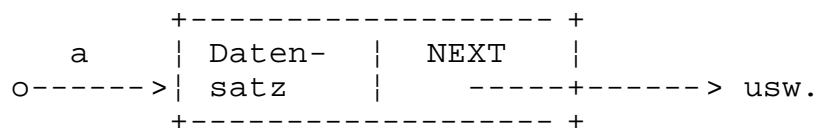
wenn noch kein Hinweis auf a^ durch NEW(a) geschaffen wurde oder wenn vorher die Anweisung a:= NIL erfolgte!

2. Listen

2.1 Vereinbarung einer linearen Liste

Bez.: Eine lineare Liste ist eine Datenstruktur, in welcher es zu einem Datenelement genau ein nachfolgendes Element gibt, außer zu dem letzten Element.

Lineare Listen lassen sich mit Hilfe von Zeigern leicht realisieren.



Jedes Datenelement besteht aus zwei Komponenten:

1. Die erste Komponente enthält die zu speichernde Information vom Typ Datensatztyp.
2. Die zweite Komponente ist ein Zeiger zum nächsten Datenelement der Liste.

Die beiden Komponenten werden in einem Verbund mit dem Namen Elementtyp zusammengefaßt.

```
TYPE DATENSATZTYP = RECORD
    Name      : string[20];
    Vorname   : string[20];
    Weite     : integer;
END;
```

Dieser Datentyp kann je nach konkreter Problemstellung verändert werden. Die folgenden Vereinbarungen sind dann unabhängig von der speziellen Definition dieses Typs.

```
TYPE ZEIGERTYP = ^ELEMENTTYP;
ELEMENTTYP = RECORD
    DATENSATZ : DATENSATZTYP;
    NEXT      : ZEIGERTYP;
END;
```

Da lineare Listen stets den gleichen Aufbau haben, können wir die hier gewählten Bezeichnungen und Vereinbarungen immer wieder verwenden.

Für ein spezielles Problem müssen wir lediglich den Datentyp Daten in der Typvereinbarung festlegen.

Für den Zugriff auf die Liste braucht man besondere Zeiger: ANFANG ist ein Zeiger, der auf das erste Element zeigt, AKTUELL zeigt auf ein anderes Element der Liste oder ein geschaffenes Element.

Bez.: Ist der Anfangszeiger ANFANG gleich NIL, dann nennt man die Liste eine leere Liste.

Anm.: Es ist zu beachten, daß bei der Definition des Zeigertyps ZEIGERTYP der Bezugstyp ELEMENTTYP verwendet wird, obwohl er noch nicht definiert ist (rekursive Datenstruktur).

2.2 Operationen auf Listen

Um die zu Beginn aufgestellten Forderungen für ein Wettkampfprogramm zu erfüllen, werden die Daten eines Wettkampfteilnehmers also in einer linearen Liste organisiert.

Gemäß der Aufgabenstellung müssen folgende Grundoperationen programmiert werden:

- Einfügen in eine geordnete Liste
- Entfernen eines Datenelementes
- Durchlaufen und Ausgabe der Liste

Weitere Operationen sind:

- Bestimmen der Listenlänge
- Speichern der Liste auf Diskette
- Einlesen der Liste von Diskette

Bevor die Teilaufgaben gelöst werden, wird zunächst das Hauptprogramm entwickelt. Um das Programm benutzerfreundlich zu gestalten, wird zunächst mit Hilfe der Unit "menurg" ein Rollbalkenmenü erzeugt.

Die Prozedur LIESINT dient der absturzsicheren Eingabe von ganzen Zahlen.

```
program WETTKAMPF;
```

```
uses crt, turbo3, menurg;
```

```
{ Dateiname: wett.pas  }  
{ Datum      : Aug.91  }  
{ Autor      : Gk Informatik.  }  
{ Zweck      : Lineare Liste mit Zeigern  }
```

```
{ Programmbeschreibung:  }  
{ Grundoperationen auf einer linearen Liste mit }  
{ Zeigervariablen      }  
{ Globale Vereinbarungen }
```

```
TYPE WORT = STRING[20];
```

```
TYPE DATENSATZTYP = RECORD  
    Name      : WORT;  
    Vorname   : WORT;  
    Weite     : integer;  
END;
```

```
{je nach Aufgabenstellung}
```

```
TYPE ZEIGERTYP  = ^ELEMENTTYP;  
ELEMENTTYP = RECORD  
    DATENSATZ: DATENSATZTYP;  
    NEXT      : ZEIGERTYP;  
END;
```

```
VAR ANFANG, AKTUELL: ZEIGERTYP;  
{somit lineare Liste definiert}
```

```
var  vorn, na: wort;  
     ant: char;
```

```

PROCEDURE TASTE;
begin gotoxy(60,22);
    write(' TASTE! ');
    write(chr(7));
    repeat until readkey <> '';
end;

PROCEDURE LIESINT(var n:integer);
{absturzsichere Eingabe einer ganzen Zahl}

var ok:boolean;
    x,y:integer;

begin repeat {$I-}
    x:=wherex;
    y:=wherey;
    readln(n);
    ok:=ioresult=0;
    if not ok then begin
        write(chr(7));
        gotoxy(x,y);clreol;
    end;
    until ok;
    {$I+}
end; {liesint}

PROCEDURE AUSWAHLMENUE;
{Programmkopf und Hauptmenü}
var sel: byte;    {Info siehe unten}
    lines: str_array;

{Information zu unit menurg;
interface uses crt, dos;
    const max_lines=10;
    type str_array=array[1..max_lines] of string;
    procedure
centermenu(lines:str_array;nlines,f,x,y:byte;var sel:byte);

    lines      : str_array mit den Menüpunkten
nlines       : Zahl der Menupunkte
x,y          : Koordinaten der linken oberen Ecke des Menüs
sel          : Variable zur Aufnahme der gewählten Nummer}

begin {auswahlmenü}
    lines[1]:=' Eingabe eines Teilnehmers ';
    lines[2]:=' Teilnehmerliste ausgeben ';
    lines[3]:=' Teilnehmer entfernen ';
    lines[4]:=' Zahl der Teilnehmer bestimmen ';
    lines[5]:=' Liste abspeichern ';
    lines[6]:=' Liste einlesen ';
    lines[7]:=' E N D E ';
    repeat clrscr;
        highvideo;
        gotoxy(18,1);
        writeln('WEITSPRUNG-WETTKAMPF');
        gotoxy(18,2);

```

```

        writeln('=====');
        lowvideo;
        gotoxy(16,3);
        writeln('(vom Info -Kurs Steffen, 13/1, 1991)');
        centermenu(lines,7,2,16,7,sel);
        case sel of
        1: begin end;
        2: begin end;
        3: begin end;
        4: begin end;
        5: begin end;
        6: begin end;
        7: begin clrscr;
            gotoxy(20,20);
            write('Tschuesss!');
            end;
        end; {case}
    until sel=7;
end; {auswahl}
BEGIN {haupt}
    anfang:= nil; {Vorbereitung}
    auswahlmenue;
END.

```

Zur Ein- und Ausgabe eines Teilnehmers dienen die beiden folgenden Prozeduren:

```

PROCEDURE LIES(VAR A : ZEIGERTYP);
{liest die Daten eines Teilnehmers ein}
begin write('Vorname      : '); readln(a^.datensatz.vorname);
      write('Name        : '); readln(a^.datensatz.name);
      write('Weite in cm : '); liesint(a^.datensatz.weite);
end;

```

```

PROCEDURE SCHREIBE(A:ZEIGERTYP);
{schreibt die Daten eines Teilnehmers auf den Schirm}
begin with a^.datensatz do
    writeln(vorname:21,name:21,weite:5);
end;

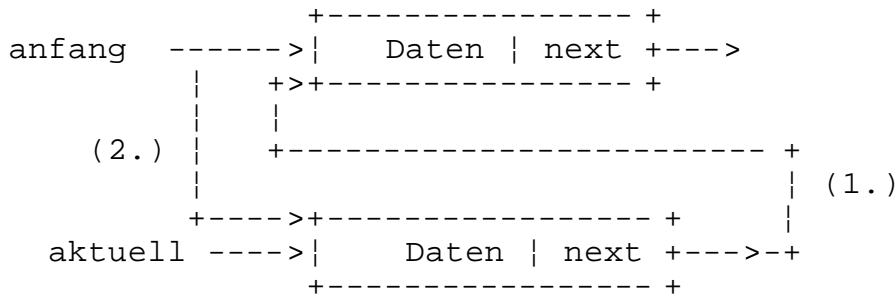
```

Einfügen in eine Liste

Die Prozedur EINSORT (vorerst EINFUEGEN) soll einen Teilnehmer in die nach den Ergebnissen sortierte Liste an der richtigen Position einfügen. Als Einstieg wird zunächst jeder Wettkämpfer an den Anfang der vorhandenen Liste gesetzt.

Dazu wird zunächst ein neues Datenelement geschaffen, auf das der Zeiger AKTUELL zeigt. Dann werden die Zeiger wie folgt "umgebogen":

```
AKTUELL^.NEXT:=ANFANG;
ANFANG := AKTUELL;
```



```
PROCEDURE EINFUEGEN(var anfang: zeigertyp);
{fügt Element am Anfang ein}
begin new(aktuell);
      lies(aktuell);
      aktuell^.next:=anfang;
      anfang:=aktuell;
end;
```

Ausgabe einer Liste

Die Prozedur LISTEAUSGEBEN kann rekursiv formuliert werden. Falls der Listenanfang nicht gleich NIL ist, wird das erste Element ausgegeben. Anschließend wird das gleiche Verfahren auf die Restliste angewandt.

```
PROCEDURE LISTEAUSGEBEN(a:zeigertyp);
begin if a <> nil then
      begin schreibe(a);
            listeausgeben(a^.next);
      end;
end;
```

Entfernen eines Elementes aus einer Liste

Der zu löschende Datensatz wird durch den Namen und den Vornamen bestimmt. Auch das Löschen kann wieder rekursiv formuliert werden. Der Zeiger vor dem zu löschenden wird auf das Element nach diesem verbogen:

```
ANFANG:=ANFANG^.NEXT
```

Damit der Speicherplatz des gelöschten Elementes wieder frei wird, läßt man einen Hilfszeiger darauf zeigen und mit DISPOSE(HILF) den Speicherplatz wieder freigeben.

```

PROCEDURE LOESCHE(var a:zeigertyp; v,n: wort);
{V : Vorname, N: Name des zu löschenden Teilnehmers}
var hilf: zeigertyp;
BEGIN if a = nil then writeln(' nicht gefunden!')
      else
        if (a^.datensatz.vorname=v) and
            (a^.datensatz.name=n) then
          begin hilf:=a;
                a:=a^.next;
                dispose(hilf);
                writeln(' Datensatz entfernt!');
          end
        else loesche(a^.next,v,n);
      END;

```

Im Hauptprogramm werden der Vorname und Name des Teilnehmers eingelesen und als Parameter an die Prozedur übergeben.

Bestimmung der Listenlänge

Die Bestimmung der Listenlänge erfolgt durch eine rekursive Funktion. Die Länge der Liste ist gleich Länge der Restliste plus 1!

```

FUNCTION LISTENLAENGE(a:zeigertyp):integer;
begin if a = nil then listenlaenge:=0
      else listenlaenge:=1+listenlaenge(a^.next);
end;

```

Abspeichern einer Liste

Das Abspeichern aller Teilnehmer kann analog zum Ausgeben aller Teilnehmer erfolgen. Nur werden die Daten nicht auf den Bildschirm, sondern in eine Datei gelenkt.

Möglich ist aber auch eine iterative Lösung. Sie sieht folgendermaßen aus:

```

PROCEDURE LISTESPEICHERN(a:zeigertyp);
var datei: file of datensatztyp;
    dateiname: string;
begin clrscr;
      writeln(' Liste abspeichern');
      writeln;
      write(' Dateiname : '); readln(dateiname);
      assign(datei,dateiname);
      rewrite(datei);
      while a <> nil do
        begin write(datei,a^.datensatz);
              a:=a^.next;
        end;
      close(datei);
      writeln(' Liste abgespeichert!');
      taste;
end;

```

Einlesen einer Liste

Das Einlesen der Liste erfolgt hier ebenfalls iterativ. Dabei ist folgendes zu beachten: Beim Lesen wird der zuerst gespeicherte Datensatz, also der Listenanfang zuerst eingelesen. Baut man die Liste nun wie beim einfachen Anfügen auf, so steht der ursprüngliche Listenanfang am Ende der Liste!

Es bietet sich daher an, die Datensätze der Reihe nach einzulesen und in der richtigen Reihenfolge in die Liste einzusortieren.

Dazu wird die rekursive Prozedur EINSORT definiert. Ist das Ergebnis des neuen Wettkämpfers besser als das desjenigen Wettkämpfers, auf den der Zeiger ANFANG zeigt, so wird der neue Wettkämpfer vorne angefügt. Andernfalls wird das Verfahren mit der Restliste wiederholt! Der Fall der leeren Liste muß gesondert betrachtet werden, da hier noch kein Ergebnis zum Vergleichen vorhanden ist.

```
PROCEDURE EINSORT(var anfang,aktuell: zeigertyp);
{sortiert nach Weitsprungergebnissen ein}
begin if anfang = nil then
    begin anfang:= aktuell;
          aktuell^.next:=nil;
    end
    else if
aktuell^.datensatz.weite>anfang^.datensatz.weite
        then begin aktuell^.next:=anfang;
                anfang:=aktuell;
            end
        else einsort(anfang^.next, aktuell);
end;    {einsort}
```

```
PROCEDURE LISTELESEN(var anfang: zeigertyp);
var datei: file of datensatztyp;
    dateiname: string;
```

```
begin clrscr;
    writeln('    Einlesen der Wettkampfdatei');
    writeln('    =====');
    writeln;
    write('    Name der Datei: '); readln(dateiname);
    assign(datei,dateiname);
    reset(datei);
    anfang:=nil;
    while not eof(datei) do
    begin new(aktuell);
        read(datei,aktuell^.datensatz);
        schreibe(aktuell);
        einsort(anfang, aktuell);
```

```

        end;
        close(datei);
        writeln('      FERTIG!');
        taste;
end;

```

Eine Prüfung, ob die Datei mit dem eingegebenen Namen existiert, erfolgt hier nicht, bietet sich aber für den praktischen Einsatz eines solchen Programms an.

Unser Programm erfüllt nun alle gestellten Anforderungen. Hier noch einmal die Prozedur AUSWAHLMENÜ, in der die beschriebenen Prozeduren aufgerufen werden.

```

PROCEDURE AUSWAHLMENUE;
{Programmkopf und Hauptmenü}

var sel: byte;   {Info siehe oben}
    lines: str_array;

begin {auswahlmenü}
    lines[1]:= ' Eingabe eines Teilnehmers ';
    lines[2]:= ' Teilnehmerliste ausgeben ';
    lines[3]:= ' Teilnehmer entfernen ';
    lines[4]:= ' Zahl der Teilnehmer bestimmen ';
    lines[5]:= ' Liste abspeichern ';
    lines[6]:= ' Liste einlesen ';
    lines[7]:= ' E N D E ';
    repeat clrscr;
        highvideo;
        gotoxy(18,1);
        writeln('WEITSPRUNG-WETTKAMPF');
        gotoxy(18,2);
        writeln('=====');
        lowvideo;
        gotoxy(16,3);
        writeln('(vom Info-Kurs Steffen, 13/1, 1991)');
        centermenu(lines,7,2,16,7,sel);
        case sel of

1: begin
        {erste Version: nur einfügen
        clrscr;
        writeln('      Neuer Teilnehmer');
        einfuegen(anfang);
        end;}

        {endgültig: gleich einsortieren}
        repeat
            clrscr;
            writeln('      Neuer Teilnehmer');
            writeln;
            new(aktuell);
            lies(aktuell);
            einsort(anfang,aktuell);
            writeln;

```

```

        write('    Noch ein Teilnehmer? (J/N) : ');
        readln(ant);
    until ant in ['n','N'];
    end;

2: begin clrscr;
    writeln('    aktuelle Liste');
    writeln;
    listeausgeben(anfang);
    taste;
    end;

3: begin clrscr;
    writeln('    Entfernen eines Datensatzes');
    writeln;
    write('    Vorname : ');readln(vorn);
    write('    Name      : ');readln(na);
    loesche(anfang,vorn,na);
    taste;
    end;

4: begin clrscr;
    writeln('Teilnehmerzahl:',listenlaenge(anfang):8);
    taste;
    end;

5: begin listespeichern(anfang);
    end;

6: begin listelezen(anfang);
    end;

7: begin clrscr;
    gotoxy(20,20);
    write('Tschuesss!');
    end;
end; {case}
until sel=7;

end; {auswahl}

```

3. Stapel

Ein unordentlicher Sachbearbeiter in einem Büro mag die eingehenden Anträge auf einen Stapel legen und die Anträge zum Bearbeiten grundsätzlich von oben herunternehmen. Betrachtet man die Anträge als Datensatz, so bildet der Stapel eine geordnete Datenstruktur, bei der grundsätzlich nur auf das oberste Element des Stapels zugegriffen werden kann. Der zuletzt abgelegte Datensatz wird als erster wieder entnommen. (LIFO: last in- first out).

Die Datenstruktur Stapel (auch Keller oder Stack genannt) kann auf zwei Arten realisiert werden:

- mit einem Feld und einer Zählervariablen, die auf den obersten Datensatz zeigt (statische Datenstruktur)
- dynamisch mit Zeigervariablen

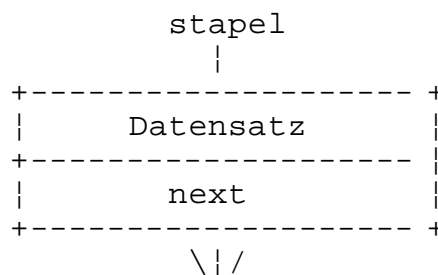
Hier soll der Stapel dynamisch mit Zeigervariablen realisiert werden. Im Grunde handelt es sich um eine lineare Liste mit eingeschränkten Zugriffsmöglichkeiten. Bei der Liste kann auf jedes Element zugegriffen werden, beim Stapel nur auf das oberste Element.

3.1 Vereinbarung eines Stapels

Zunächst wird ein Datensatztyp festgelegt. In einem einfachen Beispiel sollen nur einzelne Zeichen auf dem Stapel abgelegt werden.

```
type datensatztyp = char;
  stapeltyp = ^elementtyp;
  elementtyp = record inhalt: datensatztyp;
                  next : stapeltyp;
  end;

var stapel: stapeltyp;
    zeichen: datensatztyp;
```



3.2 Operationen auf einem Stapel

1. Erzeugung eines leeren Stapels

Dazu wird lediglich der Zeiger, der auf das oberste Element zeigt, auf nil gesetzt.

```
PROCEDURE INITIAL(var oben:stapeltyp);
begin oben:=nil;
end;
```

2. Auflegen eines Datensatzes auf den Stapel

Dies entspricht dem Anfügen eines Datensatzes an den Anfang einer linearen Liste.

```
PROCEDURE AUFLEGEN(var oben:stapeltyp; i:datensatztyp);
{legt Element auf den Stapel}
var hilf: stapeltyp;
begin new(hilf);
      hilf^.inhalt:=i;
      hilf^.next:=oben;
      oben:=hilf;
end;
```

3. Entnahme des obersten Datensatzes

```
PROCEDURE HOLEN(var oben:stapeltyp;var i:datensatztyp);
{liest oberstes Element aus}
begin i:=oben^.inhalt;
      oben:=oben^.next;
end;
```

Die Prozedur HOLEN kann nur eingesetzt werden, wenn der Stapel nicht leer. Zur Kontrolle wird eine bool'sche Funktion definiert.

4. Feststellen, ob der Stapel leer ist

```
FUNCTION STAPELLEER(oben:stapeltyp):boolean;
begin if oben=nil then stapelleer:=true else
      stapelleer:=false;
end;
```

Durch die Datenstruktur und die erlaubten Operationen wird ein neuer Datentyp Stapel definiert.

Das folgende Programm liest Zeichen von der Tastatur, bis RETURN gedrückt wird. Anschließend werden alle Zeichen in umgekehrter Reihenfolge wieder ausgegeben. Das Programm legt die Zeichen lediglich auf einen Stapel und liest den Stapel anschließend wieder aus.

```

program stapeldemo;
{demonstriert Datentyp stapel mit Zeigervariablen}
uses crt;

type ... (siehe oben)

begin clrscr;
  writeln('      DEMO-Programm zum Datentyp Stapel');
  writeln('      =====');
  writeln('      vom Info-Kurs 13/1 , Steffen');
  writeln;
  initial(stapel);
  writeln('      Bitte Zeichen eingeben! Abschluß mit
Return!');
  repeat zeichen:=readkey;
    write(zeichen);
    auflegen(stapel,zeichen);
  until zeichen=chr(13);
  writeln;
  writeln('      Zeichenfolge wird umgekehrt:');
  writeln;
  {jetzt auslesen}
  while not stapelleer(stapel) do
  begin holen(stapel,zeichen);
    write(zeichen);
  end;
end.

```

Mit Zeigervariablen lassen sich weitere Datenstrukturen von teilweise hoher Komplexität aufbauen. Zur Arbeit mit dynamischen Datenstrukturen sollten nur wenige, gut getestete Prozeduren, etwa zum Löschen und Einfügen, benutzt werden.

4. Weitere dynamische Datenstrukturen

4.1 Schlange

Bei diesem Datentyp werden Datensätze nur vorne angefügt und können nur am Ende wieder entnommen werden.

4.2 Binäre Bäume

Bei der linearen Liste könnte der Verbund auch zwei Zeigervariable enthalten, d.h. jedes Datenelement kann auf zwei weitere Elemente zeigen. Damit ließen sich zum Beispiel Stammbäume realisieren. Jedes Element zeigt auf zwei weitere Elemente (Vater und Mutter).

Im folgenden Demo-Programm wird eine baumartige Datenstruktur vereinbart.

```
program baumdemo;
{vereinbart Baumstruktur}

type datensatztyp = record
    name: string[20];
    titel: string[40];
    jahr: integer;
end;

baumtyp = ^elementtyp;
elementtyp = record
    datensatz: datensatztyp;
    links: baumtyp;
    rechts:baumtyp;
end;

var baum, aktuell: baumtyp;
procedure initial(b:baumtyp);
begin b:=nil;
end;
procedure lies(var b:baumtyp);
begin new(b);
    write('Name: ');readln(b^.datensatz.name);
    write('Titel: ');readln(b^.datensatz.titel);
    write('Jahr : '); readln(b^.datensatz.jahr);
end;

begin initial(baum);
    lies(aktuell);
end.
```

Zum Arbeiten mit solchen baumartigen Datenstrukturen werden vor allem Prozeduren zum Einfügen und zum Löschen von Datenelementen benötigt.