```
program Struktur (input, output); {Beispielprogramm zur Darstellung aller möglichen
Strukturelemente unter Anwendung der Stilrichtlinien des Kurses 1613 -- nicht benötigte
können grundsätzlich weggelassen werden}
  const
  KONSTANTE = 5; {Bezeichner: nur unmodifizierte Buchstaben und nach einem Buchstaben auch
  Zahlen}
  PI = 3.141592654;
  FELDGROESSE = 5;

  type
  tTyp = 1..MAXINT; {positive Ganzzahl --> Ausschnittstyp}
  tAufzaehlungstyp = (AufzaehlungsElement1, AufzaehlungsElement2, AufzaehlungsElement3);
  {wie char: succ und pred funktionieren, daher auch < / >}
  tAusschnittVonAufzaehlung = AufzaehlungsElement2..AufzaehlungsElement3;

  tArrayIndex = 1..FELDGROESSE; {Alternative: jeder Ausschnittstyp oder Aufzählungstyp}
  {Konstante statt absoluter Zahl, um einfache Anpassung oder einfachen Abruf in
  for-Schleife zu ermöglichen}
  tArray = array [tArrayIndex] of integer; {Größe durch Typ definiert, dynamische Größe
  nicht möglich, dafür einfach größeres Feld, dass nicht vollständig verwendet wird;
  Alternative für Definition mit implizitier Typdefinition (im Kurs verpönt): array[1..5]}
  tMultidimensionsArray = array[1..5, 1..6] of integer;

  {Verbundtyp (Typ mit unterschiedlichen Feldarten)}
  tStunde = 0..23;
  tMinSek = 0..59;
  tZeit = record
    h : tStunde;
    m,
    s : tMinSek
  end; { tZeit; Aufruf wie bei Objektorientierten Sprachen: Zeit.h, wenn Subtyp (weiterer
  record im record) z. B. Zeit.h.stelle1 usw. }

  {Zeiger (Einheit III)}
  tRefTyp = ^tListeMitRef; {^ (eigentlich Pfeil nach oben ("Dach"), gibt es unter Windows
  aber nicht), dann Typ, auf den der Zeiger zeigt; muss, wenn dort wieder verwendet, schon
  vor Record, der als Typ verwendet wird (s. u.) definiert werden, wirft also keinen Fehler
  auf, wenn Record-Typ noch nicht definiert wurde}
  tListeMitRef = record
    info : integer;
    next: tRefTyp; {zeigt auf nächstes Listenelement}
    {wenn statt next linkerNachfolger und rechterNachfolger entsteht Binärbaum (Einheit IV),
    der aber keine neue Programmstrukturdefinition enthält und daher hier nicht behandelt
    wird}
  end; {tListeMitRef}

  var
  variable1,
  variable2 : integer; {erlaubte Operationen: +, -, *, div, mod; / führt zu impliziter
  Umwandlung in real; Standardfunktionen: abs, sqr, succ, pred (Vor- bzw. Nachfolger, wie
  bei allen linearen Typen), odd (gerade Zahl? als boolean)}
  {Vergleichsoperatoren: =, <> (ungleich; Unterschied zu moderner Programmiersprache), <,
  <=, >, >=}
  variable3 : real; {erlaubte Operationen: +, -m *, /; Standardfunktionen: abs, sql, sin,
  cos, arctan, exp, ln, sqrt}
  variable4 : char; {Standardfunktionen: ord liefert Ordinalzahl und chr das Zeichen aus
  Ordnalzahl; succ, pred (da linearer Typ), daher auch Vergleiche erlaubt (wirken wie ord)}
```

```
  variable5 : string; { nur in einfachem Anführungszeichen bei Zuordnung; nicht in
  Standard-Pascal, aber im Kurs erlaubte Ausnahme, da in allen Implementierungen (als
  String-Array --> Unterschied zu moderner Programmiersprache: kurze Maximallänge (meist 255
  Zeichen, also array [1..255] of char)) }
  variable6 : boolean; {erlaubte Operationen: and, or, not}
  variable7 : tTyp;
  ListeMitRefAnfangZeiger,
  RefNeu : tRefTyp; {Zeiger (Einheit III)}

  function zeichenkette (zahl : integer; zeichen : char) : string; { Kurs will Kommentierung
  des Funktionsziels }
    var
    i : integer; {sonstige lokale Vereinbarungselemente möglich}

  begin
   zeichenkette := ''; {Funktionsergebnis wird durch Zuweisung an Funktionsnamen definiert,
   Änderung möglich (nicht wie return in moderneren Sprachen)}
   for i := 1 to zahl do
    zeichenkette := zeichenkette + zeichen;
  end; {Kurs verlangt Kommentar zeichenkette}

  {Prozeduren (Einheit III); können mit Funktionen gemischt definiert werden}
  procedure beispielprozedur (zahl: integer; var zeichenkette : string); {var für Variablen
  voranstellen, die an aktuellen Parameter gekoppelt sind (Veränderung wirkt auch für
  aktuellen Parameter an Aufrufstelle); Kurs verlangt als "ParaPascal" Unterscheidung nach
  in, inout und out statt dem Schlüsselwort var; ParaPascal soll aber nicht kompiliert
  werden (da unmöglich) --> Kurs will, dass zwar var verwendet wird, aber in, io und out dem
  Namen des formalen Parameters vorangestellt werden und auch so verwendet werden (in wird
  nur gelesen, io ermöglicht alles was man will und out wird am Ende einmal zugewiesen, aber
  kann nicht gelesen werden)}
    var
    i : integer; {sonstige lokale Vereinbarungselemente möglich}

  begin
   for  i := 1 to zahl do
    zeichenkette := zeichenkette + ':';
  end; {Kurs verlangt Kommentar beispielprozedur}

begin
  variable5 := {sic} 'Hallo! Viel Spaß beim Programmieren!';
  writeln (variable5);
  writeln ('Variable 1 eingeben: ');
  readln (variable1); {ohne ln kein Zeilenumbruch vor Abfrage, ansonsten gleiches Verhalten;
  mehrere Variablen können wie bei Ausgabe mit Trennung durch Komma eingelesen werden}
  writeln ('Wert Variable 1: ', variable1); {ohne ln kein Zeilenumbruch nach Ausgabe,
  ansonsten gleiches Verhalten}
  write ('Formartiert: ', variable1:6); {Variable:Feldbreite:Nachkommastellen bei real (wenn
  nicht gesetzt Angabe als Exonentialdarstellung mit allen Stellen)}

  if variable1 > 0 then
  begin {begin...end ersetzt abgerundete Klammern aus modernen Sprachen}
   writeln('Variable 1 ist positiv');
   write('wirklich!') {vor End kein Semikolon, da dieses nur der Trennung dient, aber
   unschädlich, da dann einfach leere Anweisung am Ende (; wie , in Aufzählungen)}
  end {wichtig: kein Semikolon, da sonst if-Anweisung abgeschlossen}
  else
   writeln('Variable 1 ist negativ');
```

```
writeln;{Leerzeile ausgeben}

{Laufanweisung (for-Schleife)}
for variable2 := 1 to 5 do
 write(variable2);

for variable2 := 5 downto 1 do
begin
 write(variable2);
 write(variable2)
end;

{Kopfschleife (Ausführung nur, wenn Schleifenbedingung erfüllt)}
while variable2 <> 5 do
begin
 writeln(variable2);
 variable2:=variable2+1;
end;

{Fußschleife (Ausführung einmal, danach nur, wenn Schleifenbedingung erfüllt)}
repeat {repeat...until ersetzt begin...end}
 writeln('Fuß',variable2);
 variable2:=variable2-1;
until variable2 > 2;

writeln('Zeichenkette: Welches Zeichen?');
readln(variable4);
writeln('Wie viele davon?');
readln(variable1);
writeln(zeichenkette(variable1,variable4));
beispielprozedur(variable1,variable5);
writeln(variable5);

{Zeiger (Einheit III)}
ListeMitRefAnfangZeiger := nil; {nil deutet auf Listenende hin -> leere Liste (Zeiger auf
leeres Element)}
new(RefNeu); {Speicherreservierung (wird bei diesen dynamischen Typen im Gegensatz zu
statischen Typen vom Programm selbst gemacht --> nachträgliche Änderung möglich);
Gegenstück (Speicherfreigabe): dispose(Ref); bei Löschen aus Liste müssen Verweise
angepasst werden, damit Liste nicht abgebrochen wird und verwaistes Stück existiert}
RefNeu^.info := 5; {^ am Ende des Zeigers führt dazu, dass wie Ziel behandelbar (sonst
kann man einem Zeiger nur einen Zeigerwert zuweisen und keine Werte abfragen)}
RefNeu^.next := nil; {nil deutet auf Listenende hin -> mit while Ref<>nil bei variablen
Ref (wird in Schleife immer durch das nächste Element überschrieben) lässt sich prüfen, ob
Liste fertig durchlaufen}
ListeMitRefAnfangZeiger := RefNeu;
new(RefNeu);
{neues Element an Anfang der Schleife)}
RefNeu^.info := 10;
RefNeu^.next := ListeMitRefAnfangZeiger;
ListeMitRefAnfangZeiger := RefNeu;
{...}
{beim Einfügen in Mitte der Liste muss darauf geachtet werden, den Wert des Elements, nach
dem eingefügt wird als ^.next des eingefügten Elements zu übernehmen (sonst verschwinden
Folgeelemente) und den ^.next des vorherigen Elements mit Zeiger auf das eingefügte
Element zu versehen (sonst wird eingefügtes Element nicht umfasst)) }
```

```
  writeln('erwarte end of file');{Unix: Strg+D, DOS: Strg+Z, Enter}
  writeln(eof); {eof wartet Ende der Eingabe ab und liefert dann True oder False; wenn in
  while-Schleife damit unbegrenzte Eingabe}
  {nach eof kein Eingabeauslesen mehr möglich}
end. {Struktur} {vorheriger Kommentar wird von Kurs vorgesehen, auch wenn unsinnig}
```