

# 10 Übung zu Informatik zum 21.1.2010 Blatt 10

## 10.1

```
program Pointerliste;  
type ZeigEle = ^Element; Element = Record Zahl: Integer; Next: ZeigEle  
end;  
var erstes: ZeigEle;  
function einfuegen_vor(var Liste: ZeigEle; x, y: Integer): Boolean;  
var aktuell, neu: ZeigEle; i: Integer = 2;  
begin  
  if (Liste = nil) or (y < 0) then einfuegen_vor := false else begin {  
    Wenn Liste nicht initialisiert oder vor Index 0 eingefügt  
    werden soll -> False}  
    aktuell := Liste;  
    if y = 1 then {Sonderfall: Element vor dem 1. Element einfügen ->  
      neues erstes Element}  
      begin  
        new(neu); {neues Element initialisieren & Werte zuweisen}  
        neu^.Zahl := x;  
        neu^.Next := aktuell; {nächstes Element = 1. Element alte Liste  
        }  
        liste := neu {Übernehme neue Liste}  
      end  
    else  
      begin  
        while (i < y) and (aktuell < nil) do {Gehe zu Element -> aktuell,  
        dessen Index größer als y ist, sofern vorhanden}  
        begin  
          i := i + 1;  
          aktuell := aktuell^.next {nächstes Element...}  
        end;  
        if aktuell = nil then einfuegen_vor := False else begin {Wenn  
          klein solches Element vorhanden, False zurückgeben}  
          new(neu); {sonst definiere neues Element}  
          neu^.Zahl := x;  
          neu^.next := aktuell^.next; {Zeiger des aktuellen Elementes  
            auf das nächste Element in neuem Element übernommen}  
          aktuell^.next := neu {Zeiger des aktuellen Elementes auf das  
            neue Element ändern}  
        end;  
      end;  
    end;  
  end;  
end;  
procedure einfuegen(var Liste: ZeigEle; x: Integer);  
var aktuell, Neu: ZeigEle;  
begin  
  aktuell := Liste;  
  if aktuell = nil then {Wenn Liste nicht initialisiert}  
  begin  
    new(Liste);  
    Liste^.Zahl := x;  
    Liste^.next := nil {Wert ist 1. Listenwert, kein nächstes  
    Element}
```

```

    end
else
    begin
        while (aktuell^.next <> nil) do {letztes initialisiertes
            Element der Liste ermitteln->aktuell}
            begin
                aktuell:=aktuell^.next
            end;
        new(neu); {Neues Element initialisieren}
        neu^.Zahl:=x;
        neu^.next:=nil;
        aktuell^.next:=neu {Neues Element letztem Element als
            Folgeelement zuweisen}
        end
    end;
procedure ausgeben(Liste:ZeigEle);
var aktuell:ZeigEle;i:Integer=0;
begin
    aktuell := Liste; {Listen-Pointer kopieren}
    while (aktuell <> nil) do {alle Listen-Elemente durchgehen}
        begin
            i:=i+1;
            writeln('Zahl_',i,' :_',aktuell^.zahl); {Zahlenwert aktuellen
                Listen-Elementes ausgeben}
            aktuell := aktuell^.next; {nächstes Listenelement verwenden}
        end;
    end;
begin
    einfuegen(erstes,5); {einfügen von 5 an letzte Stelle der Liste,
        hier Stelle 1. Liste wird in Prozedur initialisiert, wenn leer
        .}
    einfuegen(erstes,4);
    einfuegen(erstes,2);
    einfuegen(erstes,1);
    ausgeben(erstes); {Ausgeben der Liste an Oberfläche}
    writeln('Korrektur_einfuegen_von_3_an_3._Stelle:_',einfuegen_vor(
        erstes,3,3)); {einfügen einer 3 an der 3. Stelle, Ausgabe, ob
        Einfügen erfolgreich (TRUE)}
    writeln('Korrektur_einfuegen_von_-1_an_7._Stelle:_',einfuegen_vor(
        erstes,-1,7)); {Versuch, -1 an 7. Stelle einzufügen.
        Kontrollversuch (False), da nur 6 Stellen vorhanden}
    ausgeben(erstes); {Ausgabe korrigierten Wertes}
readln
end.

```

## 10.2

Das Programm gibt aus:

```
199
191
331
333
333
343
344
```

Zu Beginn: Die Integer-Werte, auf die  $i$ ,  $j$  und  $k$  zeigen werden je auf 9 gesetzt.

199: Der Prozedur eins werden die durch  $i$ ,  $j$  und  $k$  angegebenen Integer-Werte übergeben. Diese nimmt die erste übergebene Variable (Speicherstelle) und die Werte der beiden anderen Variablen (Kopien) und setzt diese 3 Parameter auf 1. Damit wurde von den der Prozedur angegebenen Variablen nur die erste geändert. Die anderen beiden bleiben außerhalb der Prozedur eins gleich.

191: Der Prozedur eins werden  $k^{\wedge}$ ,  $j^{\wedge}$  und  $j^{\wedge}$  (verwiesene Integer-Werte) übergeben. wie für "199" erklärt, wird nur die erste Variable außerhalb von eins geändert (hier  $k$ ), was bei der Ausgabe  $ijk$  den Wert 191 ergibt.

331: Der Prozedur zwei werden die Zeiger  $i$ ,  $j$  und  $k$  übergeben. Diese ruft eine interne Prozedur drei auf, der sie  $i$ - $j$ - $k$  als  $k^{\wedge}$ - $i^{\wedge}$ - $j^{\wedge}$  (die verwiesenen Integer-Werte) übergibt. die Prozedur 3 nimmt eine Kopie der ersten übergebenen Variable und die Speicherstellen der anderen beiden entgegen und setzt alle auf 3. die Prozedur zwei nimmt zwar nur Kopien der Zeiger entgegen, jedoch können die durch die Zeiger bezeichneten Werte wohl auch außerhalb der Prozedur verändert werden, da die kopierten zeiger lediglich einen Verweis auf die Wert-Speicherstelle enthalten. Somit ändert sich hier die Werte, die durch die ersten beiden Zeiger, die zwei übergeben wurden, bezeichnet sind, auf 3.

333: Hier wird 2 3 mal der Zeiger  $k$  übermittelt, wodurch der entsprechende Integer-Wert quasi zweimal außerhalb und dreimal innerhalb der Prozedur drei auf 3 gesetzt wird.

333: Der Prozedur zwei werden die Zeiger als  $i$ - $k$ - $j$  übergeben. Wie vorher beschrieben werden hier die Werte  $i^{\wedge}$  und  $k^{\wedge}$  auf 3 gesetzt. Da sie schon vorher auf 3 gesetzt waren, sieht man das aber nicht.

343: Der Prozedur vier werden zwei Integer-Werte, nämlich die durch  $i$  und  $j$  bezeichneten, übergeben. Dabei nimmt die Prozedur das erste als Kopie des Wertes und das zweite als Verweis auf die Speicherstelle entgegen. Zusätzlich definiert vier intern die Variable  $i$  als integer. Alle 3 Werte werden auf 4 gesetzt. da das übergebene  $i^{\wedge}$  lediglich eine Kopie ist und das intern definierte  $i$  nur intern gilt, wird lediglich  $j^{\wedge}$  auch außerhalb der Prozedur auf 4 gesetzt. Innerhalb von view werden natürlich alle Werte (Kopien) auf 4 gesetzt.

344: Hier wird vier  $j^{\wedge}$  und  $k^{\wedge}$  übergeben, wodurch wie eben beschrieben nur  $k^{\wedge}$  außerhalb auf 4 gesetzt wird.

```

program Kleinvokalpunkter;
var f:Text;g:Text;Eingabe:Char;
begin
  Assign(f,'daten.txt'); {Eingabe-Datei zum Öffnen vorbereiten}
  Assign(g,'neudaten.txt'); {Ausgabe-Datei zum Öffnen vorbereiten}
  reset(f); {Eingabe-Datei lesen}
  rewrite(g); {Ausgabe-Datei schreiben}
  while not eof(f) do {Bis zum Ende der Datei wiederholen...}
    begin
      read(f,Eingabe); {Zeichen für Zeichen in Eingabe einlesen}
      if Eingabe in ['A','E','I','O','U','a','e','i','o','u'] then
        write(g,'.') else begin {Wenn Eingabe ein Vokal, schreibe .
          in Ausgabedatei}
          if (ord(Eingabe[/i/])>=65) and (ord(Eingabe[/i/])<=90) then
            write(g,chr(ord(Eingabe[/i/])+32)) else write(g,Eingabe[/i/])
          {Wenn Eingabe ein Großbuchstabe, also Ordinalzahl zwischen
            einschl. 65 und 90 (ASCII) ist, erhöhe ASCII-Wert um 32 für
            entsprechenden Kleinbuchstaben (97-122)
            Und Schreibe diesen in Ausgabe-Datei}
          end
        end;
      close(f); {Eingabe-Datei schließen/freigeben}
      close(g); {Eingabe-Datei schließen/freigeben}
      writeln('Text aus daten.txt wurde gelesen , Vokale in Punkte und
        Grossbuchstaben in Kleinbuchstaben umgewandelt und der Text
        anschliessend in neudaten.txt abgespeichert. ');
      writeln('Enter druecken zum beenden'); {Benutzer Erfolgsmeldung
        zurueckgeben}
    Readln
  end.

```

