

11 Übung zu Informatik zum 28.1.2010 Blatt 11

11.1

```
program Textliste;
Type Charliste = ^chra;
chra=Record Zeichen:Char; next:charliste end;
var Liste,Sortliste:Charliste; {Liste und geordnete Liste deklarieren
}
procedure eintragen(s:String); {Füllt Liste mit chars aus
Übergabestring. überschreibt alte Liste.}
var i:Integer=0;akt,neu:Charliste;
begin
new(Liste); {Initalisieren von Liste}
if length(s)=0 then Liste:=nil else {Wenn kein String übergeben,
Abbruch}
begin
Liste^.Zeichen:=s[1]; {Erstes Zeichen in Hauptzeiger-Eigenschaft
Zeiger eingegeben}
akt:=Liste; {Pointer kopieren zum Durchlaufen der Liste}
for i:=2 to length(s) do {Alle Zeichen des Strings durchgehen}
begin
new(neu);
neu^.Zeichen:=s[i];
neu^.next:=nil; {Neues Element erstellen}
akt^.next:=neu; {Element hinten anfügen}
akt:=akt^.next {Zeiger auf nun erstelltes Element setzen}
end
end
end;
procedure sortieren(Unsort:CharListe); {Füllt Sortliste mit
sortierter Liste. überschreibt Sortliste}
var prev,zwi,einf:CharListe;
begin
Sortliste:=nil; {Löschen bisheriger Sortliste}
while Unsort<>nil do {alle Elemente aus Übergabeliste durchgehen}
begin
zwi:=Sortliste; {Zeiger der Sortliste kopieren zum Durchlaufen der
bisher eingefügten Elemente}
if Sortliste=nil then {Wenn die Sortierte Liste noch leer ist,
Hauptpointer-Element ändern}
begin
new(Sortliste);
Sortliste^.next:=nil;
Sortliste^.Zeichen:=Unsort^.Zeichen; {Hauptpointer-Zeichen auf
das erste Unsort-Zeichen setzen}
Unsort:=Unsort^.next {Nächster Zeiger}
end
else {Wenn bereits Elemente hinzugefügt wurden}
begin
prev:=nil;
while (zwi<>nil) and (ord(Unsort^.Zeichen) > ord(zwi^.Zeichen)) do
begin
prev:=zwi;
```

```

    zwi:=zwi^.next {Gehe solange die Pointer in der unsortierten
        Liste entlang, bis das Zeichen-Element
        des Pointers größer als das Element des aktuell einzufügenden
        Zeichens. merke dabei zwi als Pointer
        auf das aktuelles Element, das größer als das einzufügende und
        prev als pointer auf den vorherigen Pointer}
    end;
if prev=nil then {Wenn prev nicht existiert, ist das einzufügende
    Zeichen kleiner (alphabetisch) als das 1.
    Zeichen der bisherigen SortListe}
    begin
    new(einf);
    einf^.next:=Sortliste;
    einf^.Zeichen:=Unsort^.Zeichen;
    Sortliste:=einf {Füge den Pointer mit dem neuen Zeichen als 1.
        Element in die sortierte Liste ein}
    end
else {Wenn das einzufügende Zeichen müsste irgendwo außer an der
    1. Stelle der sortierten Liste eingefügt werden}
    begin
    new(einf);
    einf^.next:=zwi; {Gehe vom einzufügenden Wert per Pointer auf den
        Wert, dessen Zeichen größer ist}
    einf^.Zeichen:=Unsort^.Zeichen;
    prev^.next:=einf {Gehe vom vorangegangenen Wert, also jener, der
        gerade noch ein kleineres Zeichen enthält auf das einzufügende
        , nun eingefügte Element per Pointer}
    end;
    Unsort:=Unsort^.next
    end
end
end;

function palindrom(Eingabe:String):Boolean;
var akt,durch:Charliste; Pruf:Boolean=True;i,j:Integer;
begin
    eintragen(Eingabe); {Erstelle Liste mit Zeichen aus dem auf
        Palindrom zu prüfenden String}
    akt:=Liste; {Kopiere Pointer zum durchlaufen der Zeichen des
        Eingabestrings}
    for i:=1 to trunc(length(Eingabe)/2) do {Durchlaufe die erste Hälfte
        der Zeichen des Engabestrings}
        begin
        durch:=akt; {Nicht ab Hauptpointer alle durclaufen, sondern erst ab
            akt zur Effektivität}
        for j:= 1 to length(Eingabe)-2*i+1 do
            {Durchlaufe hier für jedes Element der 1. Hälfte die Elemente der
            2. Hälfte bis zu dem Zeichen, der soweit vom Ende des Strings
            entfernt ist,
            wie das Zeichen der übergeordneten Schleife aktuell vom Anfang
            entfernt ist.
            Also für das 1. Element bei 12 Zeichen das 12. Element, beim 5.
            Element das 8. Element}

```

```

begin
  durch:=durch^.next
end;
{akt und durch sind also nun die beiden Zeichen, die in einem
  Palindrom je immer gleich sein müssen.}
if akt^.Zeichen<>durch^.Zeichen then Pruf:=False; {Wenn ungleich=>
  kein Palindrom}
  akt:=akt^.next {Nächstes Zeichen in der 1. Hälfte des Prüfstrings}
end;
palindrom:=Pruf
end;

procedure ausgabe(AusListe:Charliste); {Gebe die Liste aus. Da hier
  Zeichenketten ausgegeben werden nach Muster a-b-c-d}
var akt:Charliste;
begin
  akt:=AusListe; {Pointer kopieren zum durchlaufen der Liste}
  while(akt^.next<>nil) do {Durchlaufe alle Elemente der Liste}
    begin
      write(akt^.Zeichen, '-'); {Gebe aktuelles Zeichen aus}
      akt:=akt^.next {Nächstes Zeichen}
    end;
    write(akt^.Zeichen);
    writeln
  end;

  begin
    {Test zur Funktionalität}
    eintragen('RENTNER');{Kann "Rentner" eingetragen, ausgegeben,
      sortiert und auf Palindrom geprüft werden? (ja, ja, Palindrom)}
    ausgabe(Liste);
    sortieren(Liste);
    ausgabe(Sortliste);
    Writeln('Palindrom_('RENTNER'):', palindrom('RENTNER'));
    eintragen('QWERTZUIOPASDFGHJKLYXCVBNM'); {Kann das unsortierte
      Alphabet ausgegeben, sortiert und auf Palindrom überprüft werden?
      (ja, ja, kein Palindrom)}
    ausgabe(Liste);
    sortieren(Liste);
    ausgabe(Sortliste);
    Writeln('Palindrom_(QWERTZUIOPASDFGHJKLYXCVBNM):', palindrom('
      QWERTZUIOPASDFGHJKLYXCVBNM')));
    readln()
  end.

```

11.2

Zu beweisen:

$\{x=A, y=B\} \ x:=x-y; \text{ if } x<0 \text{ then } (y:=x+y; x:=y-x) \text{ else } (x:=x+y) \ \{x=\max\{A,B\}, y=\min\{A,B\}\}$

Aufspaltung:

$\{x=A, y=B\} \ x:=x-y \ \{R\}$

$\{R, x<0\} \ y:=x+1; x:=y-x \ \{x=\max\{A,B\}, y=\min\{A,B\}\}$

$\{R, x\geq 0\} \ x:=x+y \ \{x=\max\{A,B\}, y=\min\{A,B\}\}$

1. If-Zweig:

Damit $\{R, x\geq 0\} \ x:=x+y \ \{x=\max\{A,B\}, y=\min\{A,B\}\}$ gilt, muss gelten:

$\{R, x\geq 0\} \Rightarrow \{x=\max\{A,B\}, y=\min\{A,B\}\} [x/x+y]$

$\Rightarrow \{R\} \Rightarrow \{x+y=\max\{A,B\}, y=\min\{A,B\}, x\geq 0\}$

Damit $\{x=A, y=B\} \ x:=x-y \ \{x+y=\max\{A,B\}, y=\min\{A,B\}\}$ gilt, muss gelten:

$\{x=A, y=B\} \Rightarrow \{x+y=\max\{A,B\}, y=\min\{A,B\}, x\geq 0\} [x/x-y]$

$\Rightarrow \{x=A, y=B\} \Rightarrow \{x=\max(A,B), y=\min(A,B), x-y\geq 0\} \Rightarrow \{x=\max(A,B), y=\min(A,B), x\geq y\}$

Da $x\geq y$ ist hier bewiesen, dass $x=\max(A,B)$ und $y=\min(A,B)$ ist.

2. If-Zweig:

$\{R, x<0\} \ y:=x+y; x:=y-x \ \{x=\max\{A,B\}, y=\min\{A,B\}\}$

Aufspaltung:

$\{R, x<0\} \ y:=x+y \ \{R'\}$

$\{R'\} \ x:=y-x \ \{x=\max\{A,B\}, y=\min\{A,B\}\}$

Damit $\{R'\} \ x:=y-x \ \{x=\max\{A,B\}, y=\min\{A,B\}\}$ gilt, muss gelten:

$\{R'\} \Rightarrow \{x=\max\{A,B\}, y=\min\{A,B\}\} [x/y-x]$

$\Rightarrow \{R'\} \Rightarrow \{y-x=\max\{A,B\}, y=\min\{A,B\}\}$

Damit $\{R, x<0\} \ y:=x+y \ \{y-x=\max\{A,B\}, y=\min\{A,B\}\}$ gilt, muss gelten:

$\{R, x<0\} \Rightarrow \{y-x=\max\{A,B\}, y=\min\{A,B\}\} [y/x+y]$

$\Rightarrow \{R\} \Rightarrow \{y=\max\{A,B\}, x+y=\min\{A,B\}, x<0\}$

Damit $\{x=A, y=B\} \ x:=x-y \ \{y=\max\{A,B\}, x+y=\min\{A,B\}, x<0\}$ gilt, muss gelten:

$\{x=A, y=B\} \Rightarrow \{y=\max\{A,B\}, x+y=\min\{A,B\}, x<0\} [x/x-y]$

$\Rightarrow \{x=A, y=B\} \Rightarrow \{y=\max\{A,B\}, x=\min\{A,B\}, x-y<0\} \Rightarrow \{y=\max\{A,B\}, x=\min\{A,B\}, x<y\}$

Da $x<y$ ist hier bewiesen, dass $y=\max(A,B)$ und $x=\min(A,B)$ ist.

Programmaussage bewiesen.

11.3

Zu beweisen:

$\{x=A, y=B, y>0\} z:=0; \text{ while } x>y \text{ do } (x:=x-y; z:=z+1); y:=z \{x=A \bmod B, y=A \operatorname{div} B\}$

Aufteilung:

$\{x=A, y=B, y>0\} z:=0, \text{ while } x>y \text{ do } (x:=x-y; z:=z+1) \{R\}$
 $\{R\} y:=z \{x=A \bmod B, y=A \operatorname{div} B\}$

damit $\{R\} y:=z \{x=A \bmod B, y=A \operatorname{div} B\}$ gilt, muss gelten:

$\{R\} \Rightarrow \{x=A \bmod B, y=A \operatorname{div} B\} [y/z]$
 $\Rightarrow \{R\} \Rightarrow \{x=A \bmod B, z=A \operatorname{div} B\}$

für die while-Schleife gilt die Invariante $x:=x-y^*(x \operatorname{div} y)$. mit $(x \operatorname{div} y)=z$

Beweis für z : $\{x:=x-y^*(z)\} x:=x-y; z:=z+1 \{x:=x-y^*(z)\}$

also $\{x:=x-y^*(z)\} x:=x-y \{x:=x+y-y^*(z)\} z:=z+1 \{x:=x+y-y^*(z+1)=x+y-y-y^*z=x-y^*(z)\}$

Da die Schleife ausgeführt wird solange $x \geq y$ und $z = \text{Schleifendurchläufe}$, gilt $z = x \operatorname{div} y$.

Also muss gelten:

$\{R\} x:=x-y^*(x \operatorname{div} y) \{x=A \bmod B, z=(x \operatorname{div} y)=A \operatorname{div} B\}$

damit dies gilt, muss gelten $\{R\} \Rightarrow \{x=A \bmod B, (x \operatorname{div} y)=A \operatorname{div} B\} [x/x-y^*(x \operatorname{div} y)]$

$\Rightarrow \{R\} \Rightarrow \{x-y^*(x \operatorname{div} y)=A \bmod B, (x \operatorname{div} y)=A \operatorname{div} B, x \geq y\}$

Also gilt nun $\{x=A, y=B, y>0\} \Rightarrow \{x-y^*(x \operatorname{div} y)=A \bmod B, (x \operatorname{div} y)=A \operatorname{div} B, x \geq y\}$

wenn nun $x=A$ und $y=B$, so ist $x \operatorname{div} y = A \operatorname{div} B$.

Außerdem gilt dann $x-y^*(x \operatorname{div} y) = A \bmod B$, da $y^*(x \operatorname{div} y)$ das letzte ganzzahlige Vielfache von y ist, welches kleiner als x ist. Indem wir nun die Differenz zu x bilden, erhalten wir den Rest der Division x/y , also $x \bmod y$.

Programmaussage bewiesen!

11.4

Potenzen zu einer Basis, endend mit der Ziffer 9 ergeben als End-Ziffer entweder 1 oder 9. Dies kann damit begründet werden, dass für die Einer-Stelle der neuen Zahl bei ganzen Zahlen nur die Einer-Stellen der Zahlen, die miteinander multipliziert die neue Zahl ergeben, von Bedeutung ist. Genauer entspricht die Einer-Stelle der neuen Zahl der Einer-Stelle des Ergebnisses der Multiplikation der Einer-Stellen der beiden alten Zahlen.

Da hier Potenzen mit einer-Stelle 9 betroffen sind, wird bei der ersten Multiplikation für die neue Einer-Stelle $9 \cdot 9 = 81$ gerechnet, wobei die neue Einer-Stelle nun 1 ist, da keine anderen Zwischenergebnisse des Potenzierens für den Exponenten 2 interessant sind. Für den Exponent 3 wird nun die Einer-Stelle des Ergebnisses wieder mit 9 Multipliziert, weswegen die neue Einer-stelle nun 9 ist. Die anderen Stellen können Vernachlässigt werden. Für Exponent 4 wäre es nun wieder $9 \cdot 9 = 81$, 1 als neue Einer-Stelle.

Folglich lässt sich eine Vorschrift ableiten, nach der bei allen geraden Potenzen $9 \cdot 9$ gerechnet wird, also 1 als Einer-Stelle herauskommt, und bei allen geraden Potenzen nun $1 \cdot 9$ gerechnet wird, also 9 die neue Einer-Stelle ist.

In unserem Beispiel haben wir in der höchsten Exponent-Ebene 99^9 , was nach der eben hergeleiteten Regel durch einen ungeraden Exponenten eine 9 als neue Einer-Stelle berechnet.

Hier kann man nun einige Schritte überspringen, da alle Potenzen von Neun entweder 1 oder 9 als Einer-Stelle haben und beides ungerade ist. Darum kann für die Maßgebene Potenz mit der Basis 99999 sagen, dass hier erneut eine 9 als Einer-Stelle einen ungeraden Exponenten unterliegt, wodurch auch wieder 9 als Einer-Stelle heraus kommt.

Da wir in Deutschland bei ganzen Zahlen ohne Nachkommastellen mit der Einer-Stelle einer Zahl aufhören, sie die gesuchte letzte Ziffer 9.