FernUniversität in Hagen

Klausurvorbereitung
Kurs 1613 „Einführung in die Imperative Programmierung“

9.Januar 2016

Dirk Friedenberger

Agenda

- Schleifen + Arrays
- Zeiger + Lineare Liste
- Rekursive Funktionen + Binärbäume
- White-Box und Black-Box Testverfahren

Arrays

- Folge gleichartiger Elemente
- Element wird über Index angesprochen

```
Feld : array [0..10] of integer;
```

for-Schleife

- wenn Anzahl der Schleifendurchläufe bekannt ist

```
for i:= 0 to 10 do
  wert[i] := 0;
```

while-Schleife

- wenn erster Durchlauf von Schleifenbedingung abhängig

```
readln(wert);
{ Auf ganze Zehner aufrunden }
while (wert mod 10) <> 0 do
  wert := wert + 1;
```

repeat-Schleife

- wenn mindestens ein Schleifendurchlauf notwendig

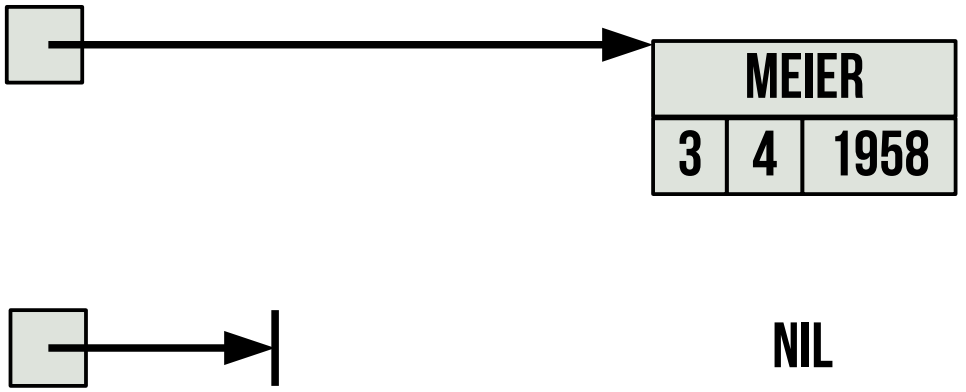
```
repeat
   { Temperatur auslesen }
   Temperatur := LiesTemperaturSensor();
until Temperatur >= 100;
writeln('das Wasser kocht');
```

Übung

HK2011 – Aufgabe 3

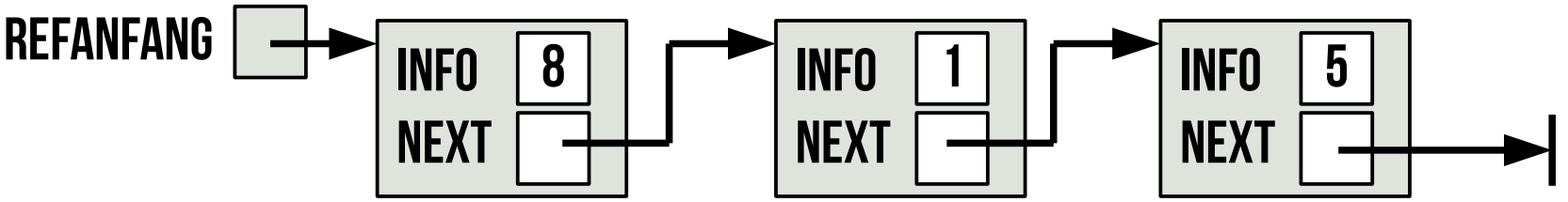
Zeiger

- Realisierung von dynamischen Datenstrukturen
- Objekte werden über Referenz angesprochen
- Speicheradresse des Objektes



Lineare Liste

- einfache dynamische Datenstruktur
- typische Operationen: (Einfügen, Suchen, Entfernen)

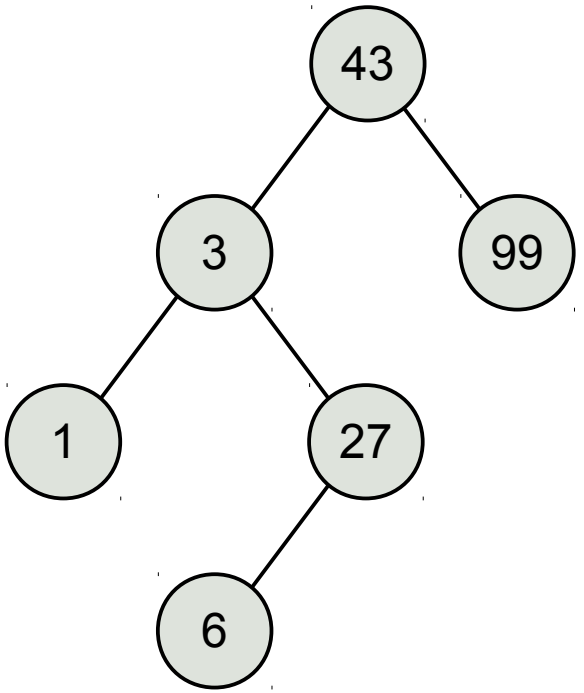


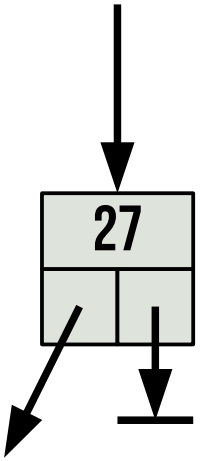
Übung

- HK 2010 Aufgabe 2

Binärbäume

- Beschleunigen der klassischen Operationen (Suchen, Einfügen, Entfernen) durch Bäume
- Speziallfall Binärbaum (jeder Knoten hat höchstens zwei Nachfolger)





Rekursive Funktionen

- Definition durch sich selbst
- Problemverkleinerung + Abbruchbedingung
- wenn interative Lösung Hilfsstapel benötigt, kann Rekursion implizit Laufzeitstack nutzen
- lokale Variablen werden pro Funktionsaufruf angelegt

$$f(n) = \begin{cases} 0 & \text{für } n = 0 \\ 1 & \text{für } n = 1 \\ f(n-1) + f(n-2) & \text{für } n > 1 \end{cases}$$

Rekursive Funktionen

- Definition durch sich selbst
- Problemverkleinerung + Abbruchbedingung
- wenn interative Lösung Hilfsstapel benötigt, kann Rekursion implizit Laufzeitstack nutzen
- lokale Variablen werden pro Funktionsaufruf angelegt

$$f(n) = \begin{cases} 0 & \text{für } n = 0 \\ 1 & \text{für } n = 1 \\ f(n-1) + f(n-2) & \text{für } n > 1 \end{cases}$$

Rekursive Funktionen

```
procedure inOrder(inRefWurzel : tRefBinBaum);
begin
  if inRefWurzel <> nil then
  begin
     inOrder(inRefWurzel^.links);
     { Betrachte 'inRefWurzel^.info' }
     inOrder(inRefWurzel^.rechts);
  end
end;
```

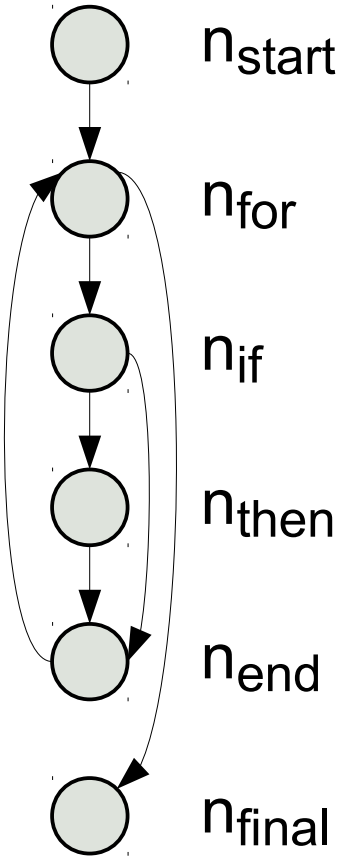
Übung

- HK 2010 Aufgabe 4

White-Box-Verfahren

- kompakter Kontrollflussgraph
- Anweisungsüberdeckung (C_0-Test)
- Zweigüberdeckung (C_1-Test)

```
for i:= 0 to 10 do
begin
   if (wert mod i) == 0 then
      wert := wert / i;
end;
```



White-Box-Verfahren

- Einfache Bedinungsüberdeckung (C_2-Test) , jede atomare Teilbedingung
- minimale Mehrfach-Bedingungsüberdeckung (jede Teilbedingung)
- Mehrfachbedingungsüberdeckung (C_3-Test) (alle Kombinationen 2^n)

```
if (i < 0) or (j < 0)then
     ...
else
     ...
```

White-Box-Verfahren

- Pfadüberdeckungstest (C_4-Test)
- Boundary-interior-Pfadtest
- Aufteilung in drei Klassen (0-,1-, n-malige Ausführung)

```
readln(wert);
{ Auf ganze Zehner aufrunden }
while (wert mod 10) <> 0 do
  wert := wert + 1;
```

Übung

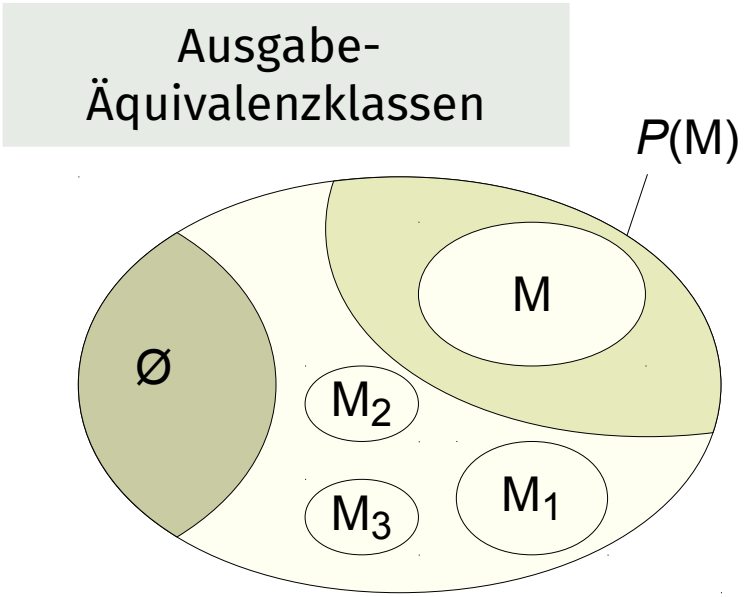
HK2009 – Aufgabe 6

Black-Box-Verfahren

- Funktionale Äquivalenzklassen

```
function Durchschnitt(A, B : tMenge): tMenge;
{ bildet den Durchschnitt von zwei Mengen }
```

$A \cap B$	$A = \emptyset$	$A \subsetneq M$ $A \neq \emptyset$	$A = M$
$B = \emptyset$			
$B \subsetneq M$ $B \neq \emptyset$	Eingabe-Äquivalenzklassen		
$B = M$			



Übung

NK2009 – Aufgabe 5

Quellen

Hans-Werner Six. "Einführung in die imperative Programmierung" Fernuniversität Hagen, 2014.

Klausuren und Musterlösungen, http://www.uniturm.de/unterlagen/einfuehrung-in-die-imperative-programmierung-6265, (abgerufen Januar 2015)