

Objekte beschreiben

Abstraktion

- von realen Gegenständen oder Konzepten
- wichtiges isolieren
- unwichtiges ignorieren
- **Bsp.:** Karteikasten mit Karteikarten

benutzerdefinierte Klasse: Klassenmodul

- **Eigenschaften** für Beschreibung des Zustands
 - ⇒ öffentliche Variablen im Klassenmodul
 - ⇒ Property Get-/Let-Prozeduren
- **Methoden** für Beschreibung des Verhaltens
 - ⇒ öffentliche Prozeduren im Klassenmodul
 - ⇒ Ereignisprozeduren
 - ⇒ Konstruktor/Destruktor

Instanzen erstellen, Objektmodelle aufbauen

Instanzen benutzerdefinierter Klassen

- 1) Objektvariable vereinbaren
- 2) Instanz erstellen und der Variablen zuweisen
- 3) Methoden und Eigenschaften benutzen

```
Bsp.: Dim neuerEintrag As cEintrag  
      Set neuerEintrag = New cEintrag  
      neuerEintrag.Name = Range("name")
```

Objektmodell

Ein **Objektmodell** beschreibt die Objekte einer Anwendung und ihre Beziehungen untereinander.

- **statisch**
 - ⇒ einzelne Objekte als Eigenschaften
 - ⇒ **Bsp.:** *“Ein Velo hat zwei Räder”*
- **dynamisch**
 - ⇒ Collection als Eigenschaften
 - ⇒ **Bsp.:** *“Eine Firma hat MitarbeiterInnen”*

Datenverwaltung

Daten verwalten heisst

- **speichern**
 - ⇒ schnell und platzsparend
- **wiederfinden**
 - ⇒ schnell und zuverlässig
- entfernen, sortieren, kopieren, ...

Beispiel: Datenfeld

- **speichern**
- **wiederfinden**
 - ⇒ direkte Suche
 - ⇒ sequentielle Suche
 - ⇒ binäre Suche

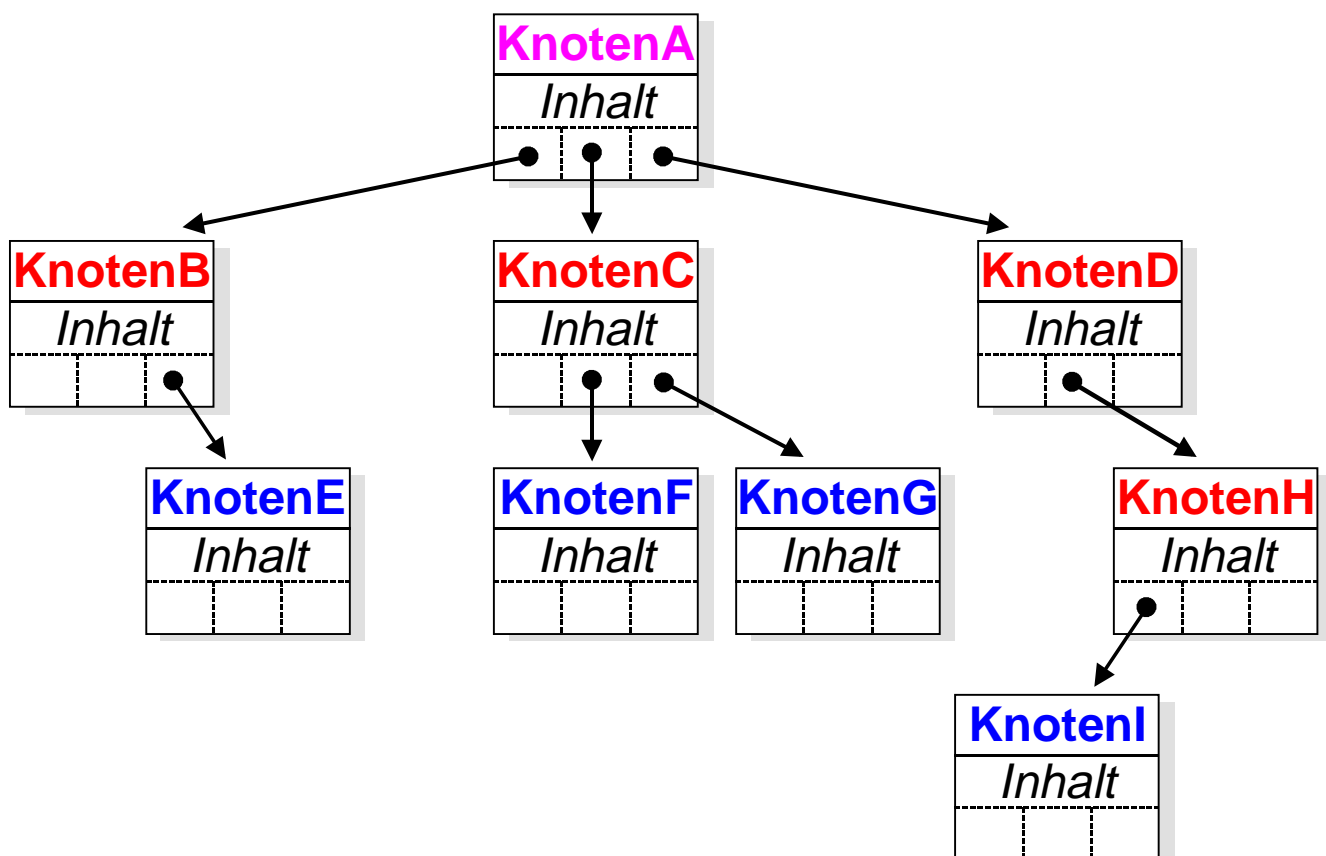
Beispiel: Bibliothek, Bücher

- **speichern**
- **wiederfinden**
 - ⇒ Katalog
 - ⇒ Inhaltsverzeichnis
 - ⇒ Index

Baum-Strukturen

Ein **Baum** ist eine kreislos verkettete Datenstruktur, wobei jedes Element auf *mehrere* Nachfolger verweisen kann.

- **verkettete** Datenstruktur
- **nicht-lineare** Datenstruktur
- die Elemente heissen **Knoten**, **Wurzel**, **Blätter**, **Eltern**, **Kinder**
- legt man für die Nachfolger eine Reihenfolge fest, so heisst der Baum *geordnet* (Achtung: geordnet \neq sortiert)
- Spezialfälle: **verkettete Liste**, **Binärbaum**



verkettete Datenstrukturen

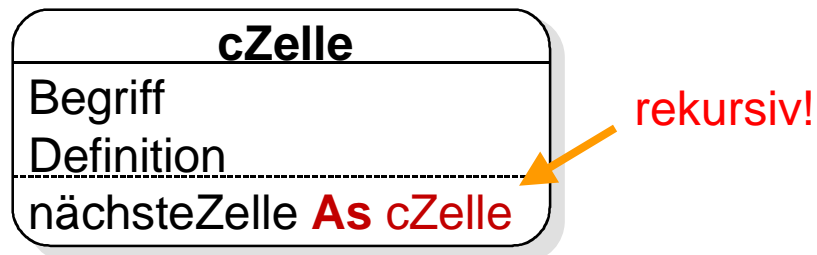
Datenstruktur

Die Elemente bestehen aus zwei Teilen:

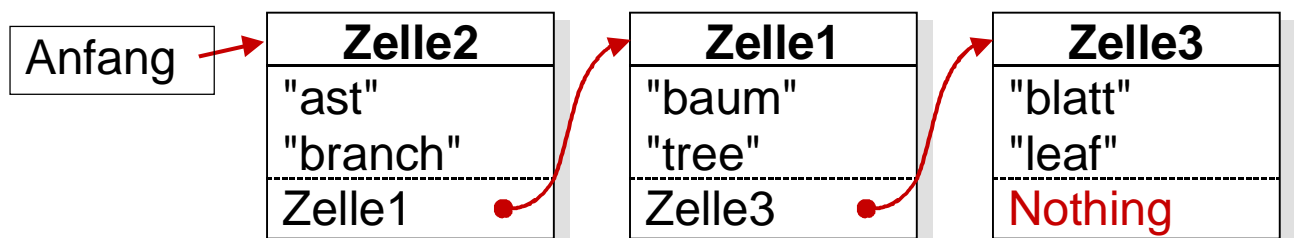
- 1) Inhalt
- 2) Verweis(e) auf andere Elemente (Verkettung)

Beispiel: verkettete Liste

Element (Zelle)



Liste



Besonderheiten

- ☺ dynamisch
- ☺ keine Reihenfolge im Speicher
- ☺ kein Kopieren von Inhalten bei Umstrukturierung (z.B. einfügen)
- ☹ keine direkte Suche

Implementation der verketteten Liste

Klassenmodul cZelle

```
Public Inhalt As String  
Public Nachfolger As cZelle
```

Codemodul

```
Private Anfang As cZelle
```

```
Private Sub Test()
```

```
Dim Zelle1 As cZelle  
Dim Zelle2 As cZelle  
Dim Zelle3 As cZelle  
Dim Zelle4 As cZelle
```

*Variablen
vereinbaren*

```
Set Zelle1 = New cZelle  
Zelle1.Inhalt = "anna"  
Set Zelle2 = New cZelle  
Zelle2.Inhalt = "carmen"  
Set Zelle3 = New cZelle  
Zelle3.Inhalt = "david"  
Set Zelle4 = New cZelle  
Zelle4.Inhalt = "beat"
```

*Elemente
erstellen*

```
Set Anfang = Zelle1  
Set Zelle1.Nachfolger = Zelle2  
Set Zelle2.Nachfolger = Zelle3
```

*Elemente
verketten*

```
fügeEin Zelle4, Zelle1
```

*Element
einfügen*

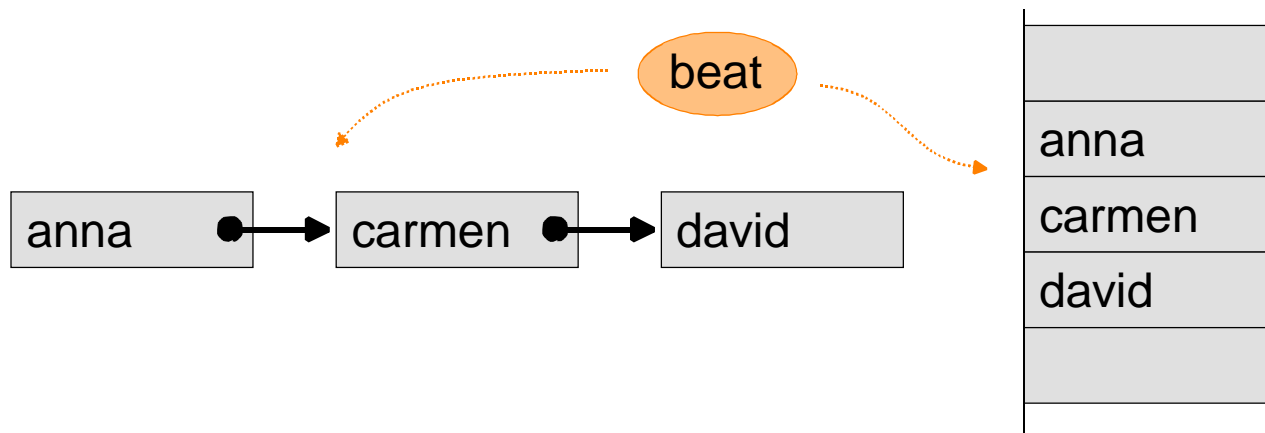
```
End Sub
```

```
Private Sub fügeEin(neu As cZelle, hinter As cZelle)
```

```
Set neu.Nachfolger = hinter.Nachfolger  
Set hinter.Nachfolger = neu
```

```
End Sub
```

Vergleich verkettete Liste - Datenfeld



verkettete Liste

Datenfeld

Einfügen

unsortiert:

😊 einfach und schnell

😞 schwierig

sortiert:

😊 einfach (und schnell)

😞 schwierig

Suchen

direkte Suche:

😞 nicht möglich

😊 schnell

sequentielle Suche:

😞 langsam

😞 langsam

Binärsuche:

😞 nicht möglich

😊 schnell

Löschen

😊 einfach und schnell,
wenn *doppelt verkettet*

😞 schwierig

Strukturen, die auch Rückwärtsverweise verwalten,
bezeichnet man als *doppelt verkettet*.

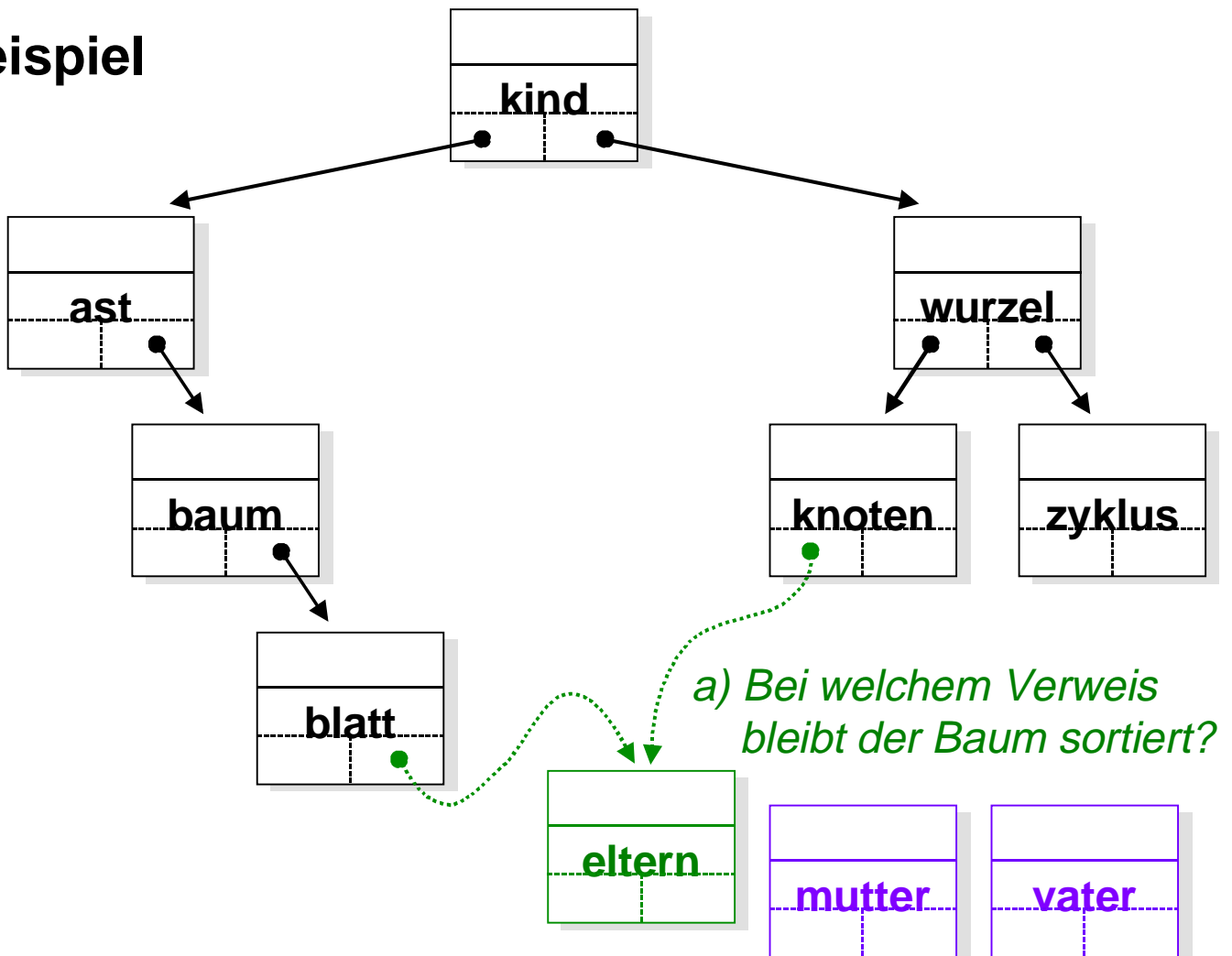
Binärbaum

Ein **Binärbaum** ist ein *geordneter* Baum, dessen Knoten *zwei* Nachfolger haben, einen *linken* und einen *rechten*.

Ein Binärbaum heisst **sortiert**, wenn *für jeden* Knoten k gilt:

1. im *linken* Unterbaum sind alle Inhalte kleiner als in k
und
2. im *rechten* Unterbaum sind alle Inhalte grösser als in k .

Beispiel



a) Bei welchem Verweis bleibt der Baum sortiert?

b) Fügen Sie die anderen zwei Knoten so ein, dass der Baum sortiert bleibt?

c) Formulieren Sie einen Einfügealgorithmus.

Einfügealgorithmus (iterativ)

Entwurf

fügeHinzult(Begriff⁽¹⁾)

FALLS Baum noch keinen Wurzelknoten hat
 erzeuge Wurzelknoten
 fülle Begriff ein.

SONST

 gehe zu Wurzelknoten

 WIEDERHOLE BIS Begriff = Begriff im Knoten

 FALLS Begriff < Begriff im Knoten

 FALLS Knoten keinen linken Nachfolger hat
 erzeuge linken Nachfolgerknoten
 fülle Begriff ein.

 gehe zu linkem Nachfolgerknoten

 SONST

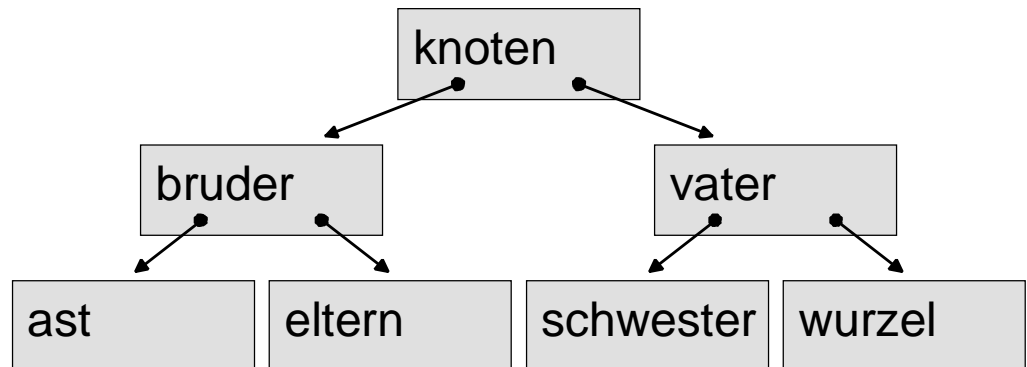
 FALLS Knoten keinen rechten Nachfolger hat
 erzeuge rechten Nachfolgerknoten
 fülle Begriff ein.

 gehe zu rechtem Nachfolgerknoten

⁽¹⁾ Begriff steht für das Begriff/Definitions-Paar

Suche im sortierten Binärbaum

ast
bruder
eltern
knoten
schwester
vater
wurzel



Beispiele

- suche 'eltern'
- suche 'vater'

Vergleich

Einfügen

Suche

Binärbaum

- 😊 schnell
- 😊 schnell wie Binär-suche im sortierten Datenfeld
(*im Idealfall !*)

verkettete Liste

- 😊 einfach und schnell
- 😞 sequentiell, langsam

Suchalgorithmus (iterativ)

Entwurf

suchelt(Begriff⁽¹⁾)

FALLS Baum keinen Wurzelknoten hat
gib **Nothing**⁽²⁾ zurück.

SONST

gehe zu Wurzelknoten

WIEDERHOLE BIS **Begriff** = Begriff im aktuellen Knoten

FALLS **Begriff** < Begriff im aktuellen Knoten

FALLS aktueller Knoten keinen **linken** Nachfolger hat
gib **Nothing**⁽²⁾ zurück.

gehe zu **linkem** Nachfolgerknoten

SONST

FALLS aktueller Knoten keinen **rechten** Nachfolger hat
gib **Nothing**⁽²⁾ zurück.

gehe zu **rechtem** Nachfolgerknoten

gib **aktuellen Knoten** zurück.

⁽¹⁾ **Begriff** steht für das Begriff/Definitions-Paar

⁽²⁾ **Nothing** wird zurück gegeben, falls gesuchter **Begriff** nicht existiert

Der “ideale Binärbaum” ?




Aufgaben


Skizzieren Sie die Baumstruktur, die entsteht, wenn Sie die folgenden Begriff in einen (noch leeren) sortieren Binärbaum einfüllen:

- a) *kind, baum, blatt, wurzel, zyklus, knoten, ast*
- b) *ast, baum, blatt, kind, knoten, wurzel, zyklus*

Fragen

- 1) Wenn Sie in den obigen Beispielen den letzten Begriff einfüllen, wie oft müssen Sie ihn dann mit bereits im Baum vorhandenen Begriffen vergleichen?
 - a)
 - b)
- 2) Wie viele Begriffe kann ein sortierter Binärbaum im Idealfall enthalten, so dass man jeden weiteren Begriff mit maximal 8 Vergleichen hinzufügen kann?


.....

- 3) Unter welchen Bedingungen tritt der *ungünstigste* Fall ein?
.....




Höhe und Ausbalanciertheit

Als **Höhe** eines Baumes bezeichnet man die Anzahl Verweise, denen man maximal folgen muss, um zu einem Blatt zu gelangen.

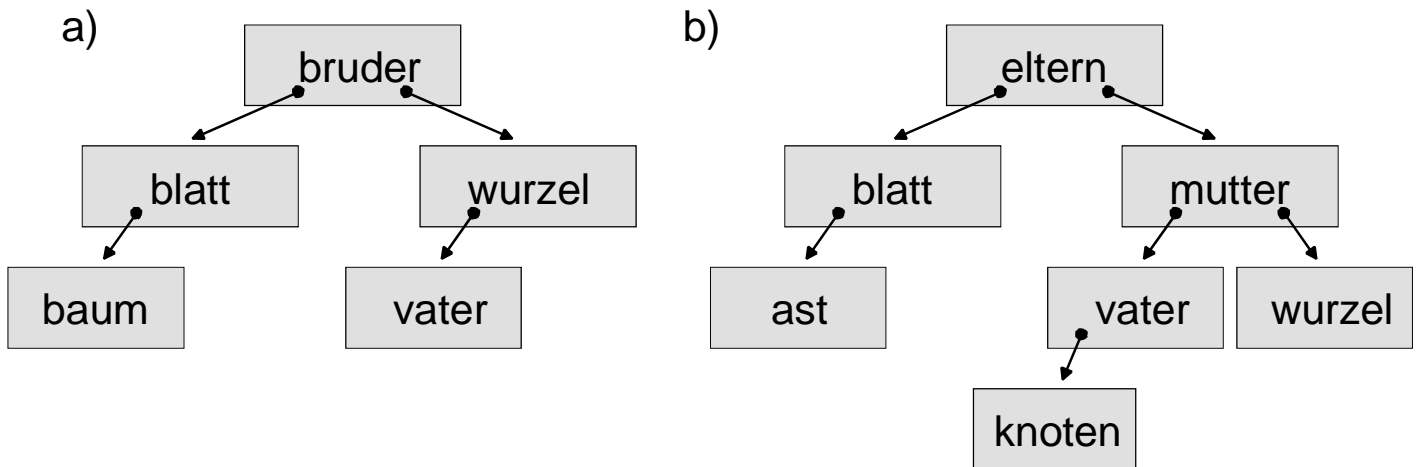
Je geringer die Höhe, desto effizienter die Algorithmen.

Die Höhe sortierter Binärbäume hängt davon ab,

- *wie viele* Knoten der Baum speichern muss und
- *wie ausbalanciert* die Teilbäume der einzelnen Knoten sind.

Ein Baum heisst **ausbalanciert**, wenn *für jeden* Knoten gilt, dass sich die Anzahl der Knoten in den beiden Unterbäumen höchstens um Eins unterscheiden.

Beispiele



Fragen:

1. Welche Höhe haben die beiden Bäume?

a)

b)

2. Welcher Baum ist sortiert, welcher nicht?

a)

b)

3. Welcher Baum ist ausbalanciert, welcher nicht?

a)

b)



rekursive Datenstrukturen

alternative Definition

Ein **Baum** ist entweder ...

1. ein Knoten ohne Nachfolger (\rightarrow Blatt) *oder*
2. ein Knoten mit **Bäumen** als Nachfolger (\rightarrow Teilbäume)

\Rightarrow rekursive Definition

\Rightarrow Auf rekursiv definierten Datenstrukturen lassen sich oft auch einfache rekursive Algorithmen formulieren.

zur Erinnerung

- Die Rekursion ist eine Form der *Wiederholung*.
- Eine alternative Form der Wiederholung ist die *Iteration*.
- Man unterscheidet zwischen *endlicher* und *unendlicher* Rekursion.
- Man unterscheidet zwischen *direkter* und *indirekter* Rekursion.

rekursive Algorithmen

Konstruktion rekursiver Algorithmen

1. **Basisfall** := Fall, der ohne rekursiven Aufruf gelöst werden kann. Der Basisfall bricht die Rekursion ab (→ Abbruchbedingungen).

z.B.: Blatt

2. **Reduktion** := reduziert die gestellte Aufgabe auf eine einfachere und lässt diese lösen (→ rekursive Aufrufe).

z.B.: Teilbaum

Damit ein Algorithmus abbricht, muss ...

- ... der Basisfall *vor* der Reduktion geprüft werden
- ... die Reduktion die gestellte Aufgabe *vereinfachen*

Einfügealgorithmus (rekursiv)

Entwurf

fügeHinzuRek(Begriff⁽¹⁾, Baum⁽²⁾)

'--- *Basisfall*

FALLS Baum leer ist
 erzeuge Wurzelknoten
 fülle Begriff ein.

'--- *Reduktionen*

SONST

FALLS Begriff < Begriff im Wurzelknoten von Baum
 fügeHinzuRek Begriff, linker Teilbaum des Baumes

FALLS Begriff > Begriff im Wurzelknoten von Baum
 fügeHinzuRek Begriff, rechter Teilbaum des Baumes

⁽¹⁾ Begriff steht wieder für das Begriff/Definitions-Paar

⁽²⁾ Teilbaum, in dem der Begriff hinzugefügt werden soll

Wann rekursiv, wann iterativ ?

Aufgabe

Wir möchten ein Wörterbuch 'Deutsch-Englisch' für Fachbegriffe implementieren.

Später möchten wir weitere Wörterbücher erstellen (z.B. 'Englisch-Deutsch').

Benutzerschnittstelle

deutsch	englisch
kind	child
<input type="button" value="füge hinzu"/>	<input type="button" value="übersetze"/>
<input type="button" value="entferne"/>	

- **füge hinzu:** Ordnet einen neuen *Eintrag* (*Begriff/Definitions-Paar*) ins Wörterbuch ein.
- **übersetze:** Gibt Definition zu einem Begriff zurück.
- **entferne:** Entfernt einen Eintrag aus dem Wörterbuch.

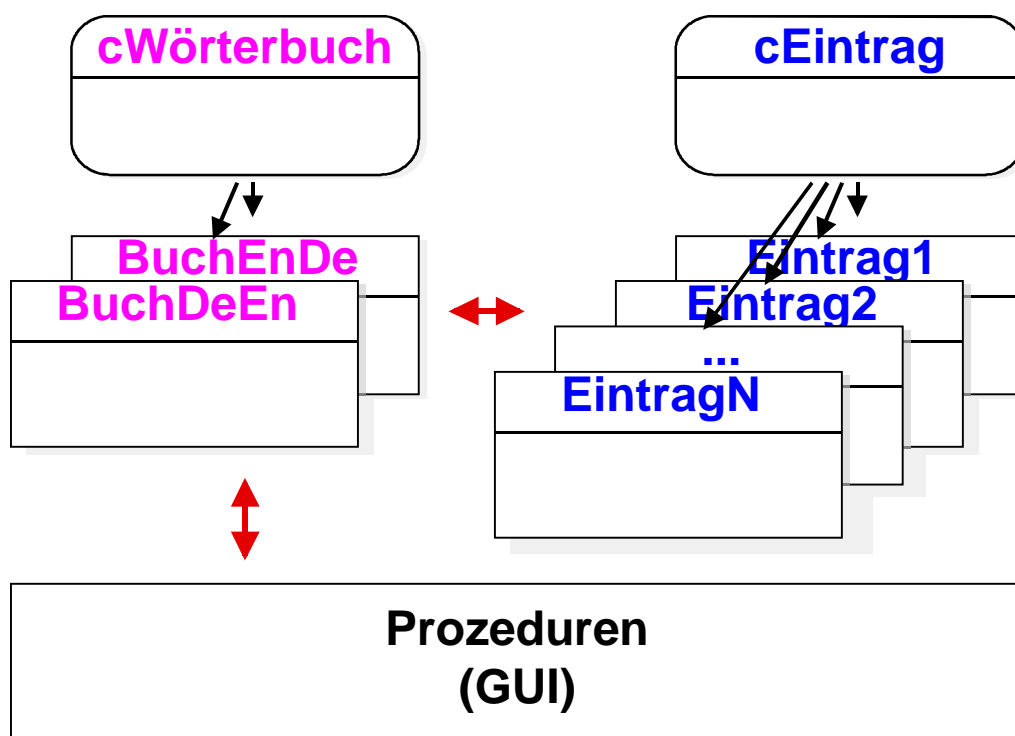
Lösungsvarianten

- ☹ Wörterbuch erstellen und später kopieren
- ☹ gemeinsamen Code in Prozeduren auslagern
- 😊 Wörterbuch in Klassenmodul implementieren

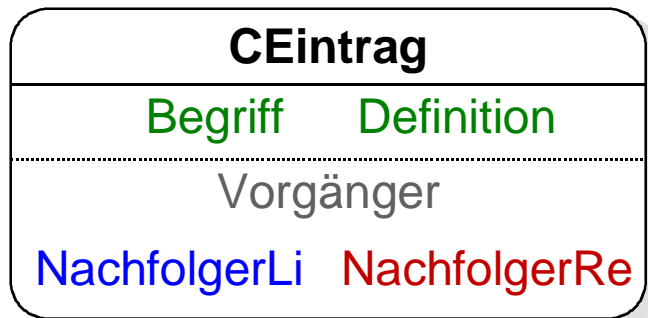
Architektur

Module

- Klassenmodul **cEintrag**
⇒ implementiert die Knoten der Bäume
- Klassenmodul **cWörterbuch**
⇒ verwaltet Wörterbuch-Einträge in einem Binärbaum
- Standardmodul Prozeduren
⇒ enthält Ereignisprozeduren der Benutzerschnittstelle



Klasse cEintrag



Eigenschaften und Methoden

Eigenschaft	Datentyp	Beschreibung
Begriff	String	Schlüssel
Definition	String	Inhalt
Vater	cEintrag	Verkettung zu Vorgänger
KindLi	cEintrag	Verkettung zu linkem Nachfolger
KindRe	cEintrag	Verkettung zu rechtem Nachfolger

Verweise auf **Vater** sind zwar redundant, vereinfachen aber die Algorithmen.

Implementation

Klassenmodul CEintrag

```
Public Begriff As String
Public Definition As String
Public KindLi As cEintrag
Public KindRe As cEintrag
Public Vater As cEintrag
```

rekursive Definition

doppelt verkettet

Klasse cWörterbuch

Eigenschaften und Methoden (in Analogie zum Collection-Objekt)

<i>Methoden</i>	<i>Collection</i>	<i>Beschreibung</i>
fügeEin(Begriff, Definition)	Add	ordnet Eintrag ein
Eintrag(Begriff)	Item	gibt Definition zurück
entferne(Begriff)	Remove	entfernt Eintrag
OErster()	For Each..Next	erster Eintrag
ONächster(Eintrag)		nächster Eintrag

- cWörterbuch sortiert die Objekte, Collection nicht
- cWörterbuch kann nur Objekte vom Typ cEintrag aufnehmen
- Wir können keine For Each...Next-Schleife implementieren

Verwenden von cWörterbuch

Beispiel

```
Dim BuchDeEn As cWörterbuch
```

```
Dim Eintrag As cEintrag
```

```
Set BuchDeEn = New cWörterbuch
```

```
BuchDeEn.fügeHinzu "baum", "tree"
```

```
BuchDeEn.fügeHinzu "ast", "branch"
```

```
BuchDeEn.fügeHinzu "kind", "child"
```

```
Set Eintrag = BuchDeEn.OErster()
```

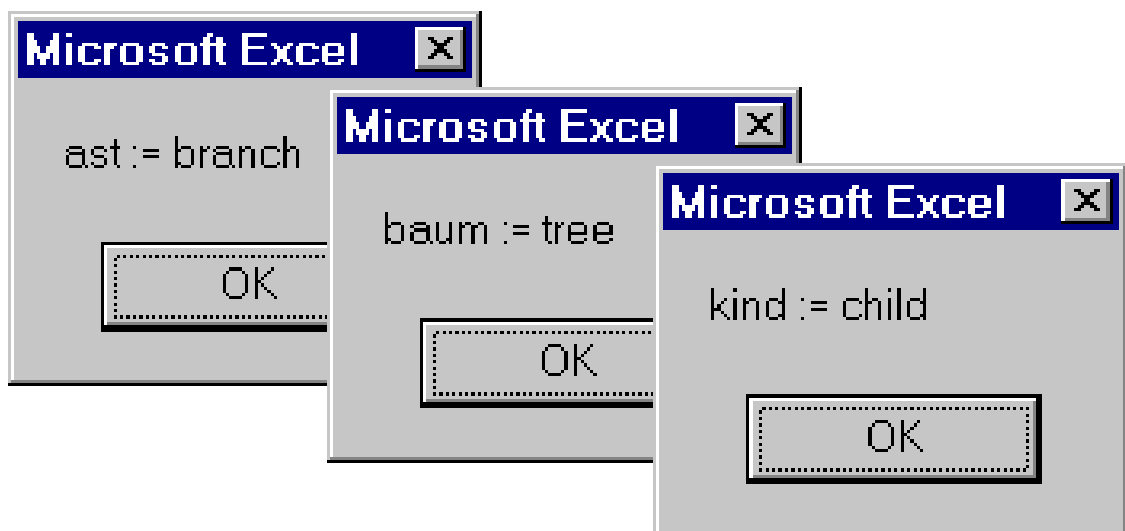
```
Do While Not Eintrag Is Nothing
```

```
    MsgBox Eintrag.Begriff & " := " & Eintrag.Definition
```

```
    Set Eintrag = BuchDeEn.ONächster(Eintrag)
```

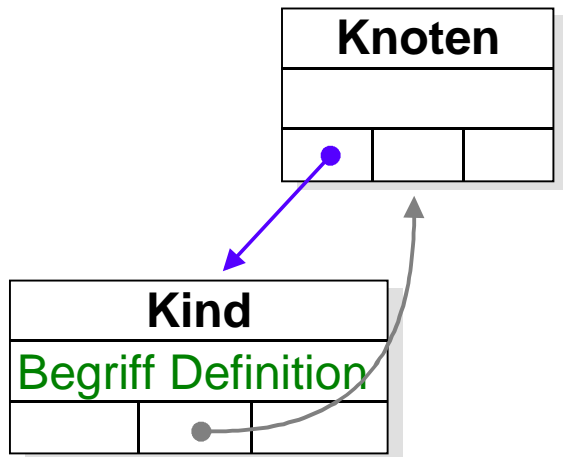
```
Loop
```

Ausgabe:

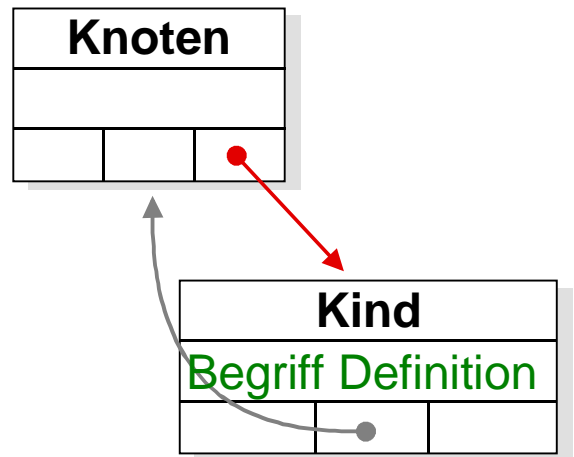


Implementation der Verkettung

erzeugeKindLi()



erzeugeKindRe()



im Klassenmodul cWörterbuch

```
Private Sub erzeugeKindLi(Knoten As cEintrag, _
    Begriff As String, Definition As String)

    Set Knoten.KindLi = New cEintrag
    Set Knoten.KindLi.Vater = Knoten

    Knoten.KindLi.Begriff = Begriff
    Knoten.KindLi.Definition = Definition

End Sub
```

```
Private Sub erzeugeKindRe(Knoten As cEintrag, _
    Begriff As String, Definition As String)

    .....

    .....

    .....

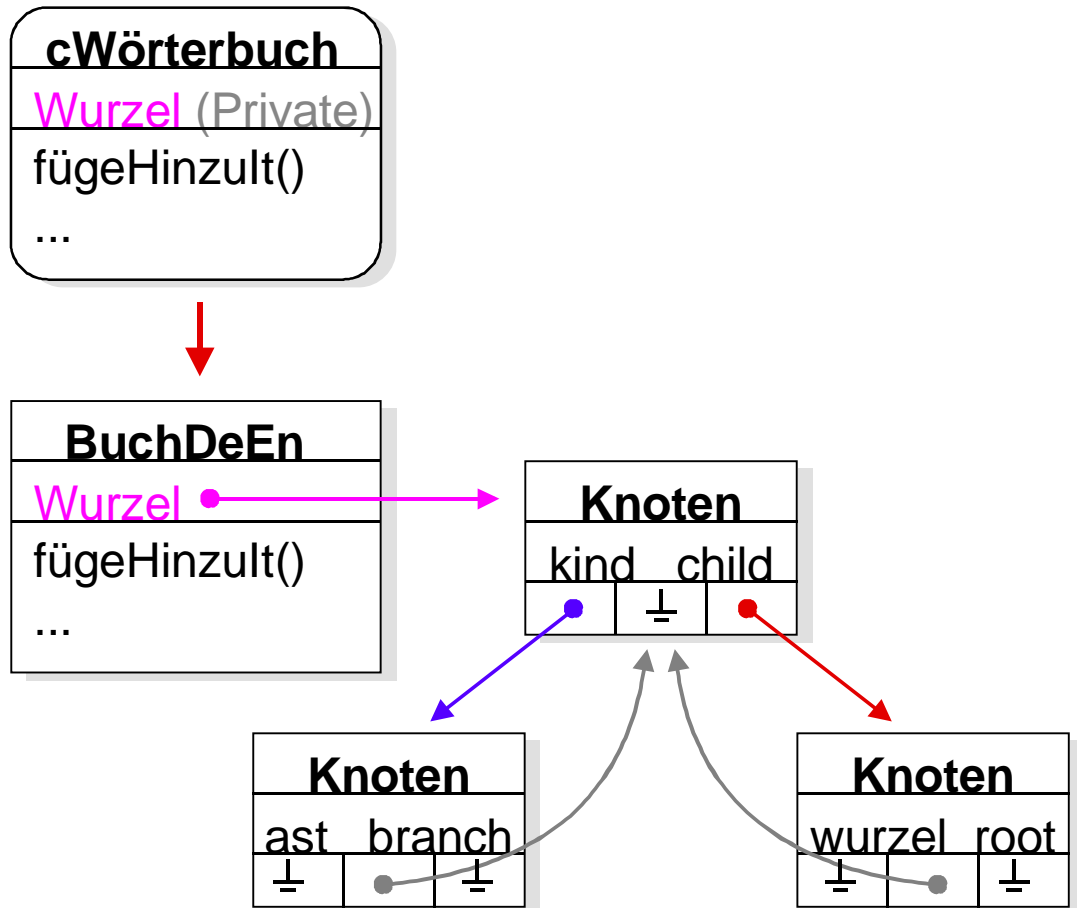
    .....

End Sub
```



Implementation der Baum-Struktur

Klasse cWörterbuch



Einfügealgorithmus (iterativ)

Implementation im Klassenmodul cWörterbuch

```
Private Wurzel As cEintrag

Public Sub fügeHinzuIt(Begriff As String, _
                      Definition As String)

    Dim Knoten As cEintrag

    If Wurzel Is Nothing Then
        Set Wurzel = New cEintrag
        Wurzel.Begriff = Begriff
        Wurzel.Definition = Definition
    Else
        Set Knoten = Wurzel
        Do Until Begriff = Knoten.Begriff
            If Begriff < Knoten.Begriff Then
                If Knoten.KindLi Is Nothing Then
                    erzeugeKindLi Knoten, Begriff, Definition
                End If
                Set Knoten = Knoten.KindLi
            Else
                If Knoten.KindRe Is Nothing Then
                    erzeugeKindRe Knoten, Begriff, Definition
                End If
                Set Knoten = Knoten.KindRe
            End If
        Loop
    End If
End Sub
```

Einfügealgorithmus (rekursiv)

Implementation im Klassenmodul CWörterbuch

```
Private Sub fügeHinzuRek(Begriff As String, _  
                        Definition As String, _  
                        Baum As cEintrag)
```

```
    'Abbruchbedingungen:
```

```
    If Wurzel Is Nothing Then
```

```
        Set Wurzel = New cEintrag
```

```
        Wurzel.Begriff = Begriff
```

```
        Wurzel.Definition = Definition
```

```
    ElseIf Begriff < Baum.Begriff _
```

```
    And Baum.KindLi Is Nothing Then
```

```
        erzeugeKindLi Baum, Begriff, Definition
```

```
    ElseIf Begriff > Baum.Begriff _
```

```
    And Baum.KindRe Is Nothing Then
```

```
        erzeugeKindRe Baum, Begriff, Definition
```

```
    'Reduktionen:
```

```
    ElseIf Begriff < Baum.Begriff Then
```

```
        fügeHinzuRek Begriff, Definition, _  
                    Baum.KindLi
```

```
    ElseIf Begriff > Baum.Begriff Then
```

```
        fügeHinzuRek Begriff, Definition, _  
                    Baum.KindRe
```

```
    End If
```

```
End Sub
```

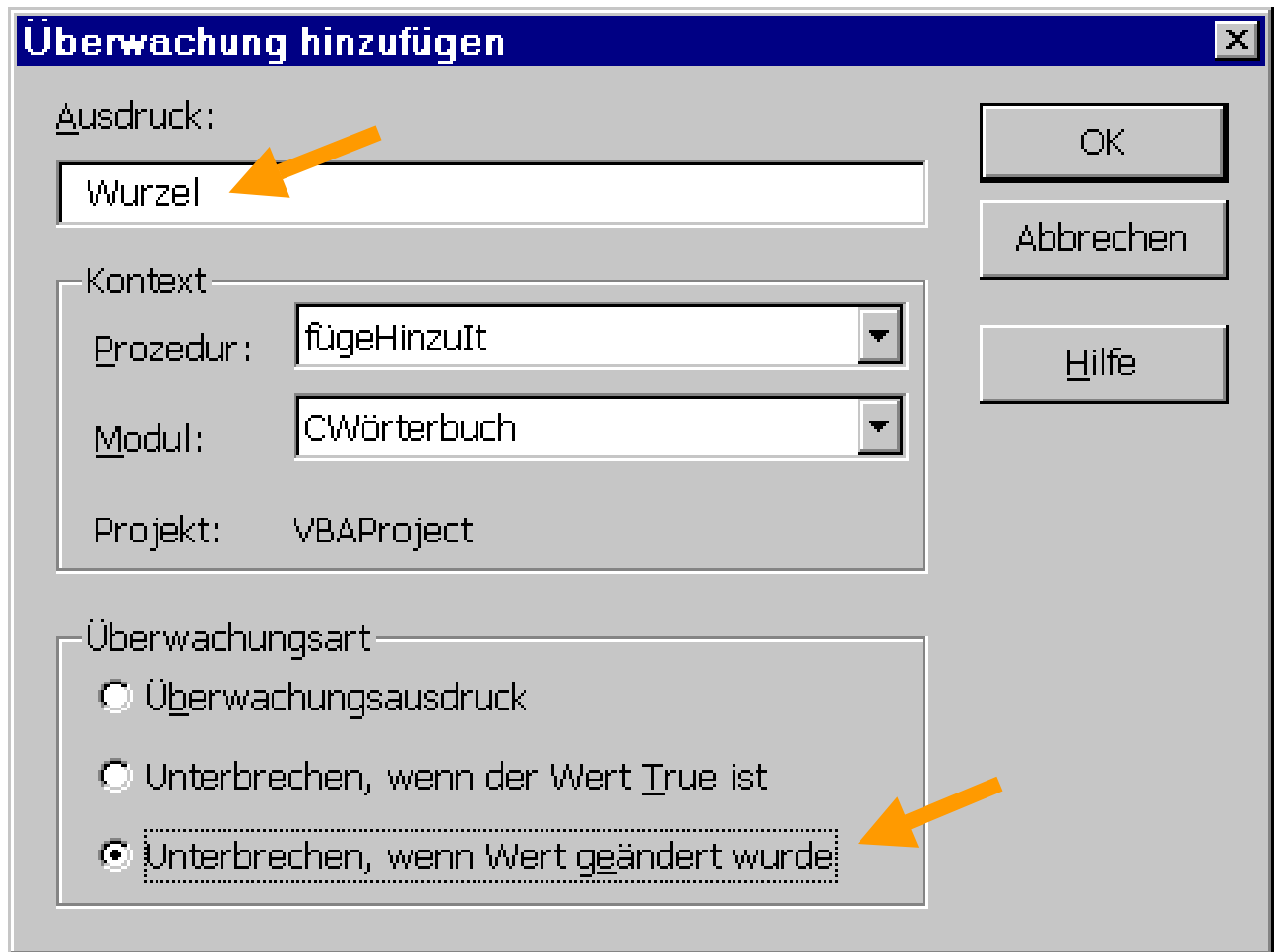
Aufruf

```
fügeHinzuRek Begriff, Definition, Wurzel
```

Test mit VBA-Überwachungsausdrücken

Überwachungsausdruck hinzufügen

- 1) Überwachung hinzufügen 
- 2) Überwachungsausdruck definieren, evtl. Haltepunkt setzen



Überwachung hinzufügen

Ausdruck:

Kontext

Prozedur:

Modul:

Projekt: VBAProject

Überwachungsart

☒ Überwachungsausdruck

☐ Unterbrechen, wenn der Wert True ist

☒ Unterbrechen, wenn Wert geändert wurde

OK

Abbrechen

Hilfe









- 3) Ausdruck im Überwachungsfenster beobachten



Ausdruck	Wert	Typ	Kontext
 Wurzel	<Nicht im Kontext>	CEintrag	CWörterbuch.fügeHinzuIt

Baumstruktur im Überwachungsfenster

nach Hinzufügen der Begriffe *kind*, *ast*, *baum*, *wurzel*

Überwachungsausdrücke	
Ausdruck	Wert
  Wurzel	
— Begriff	"kind"
— Definition	"child"
—  NachfolgerLi	
— Begriff	"ast"
— Definition	"branch"
— NachfolgerLi	Nothing
—  NachfolgerRe	
— Begriff	"baum"
— Definition	"tree"
— NachfolgerLi	Nothing
— NachfolgerRe	Nothing
—  Vorgänger	
—  Vorgänger	
—  NachfolgerRe	
— Begriff	"wurzel"
— Definition	"root"
— NachfolgerLi	Nothing
— NachfolgerRe	Nothing
—  Vorgänger	
— Vorgänger	Nothing