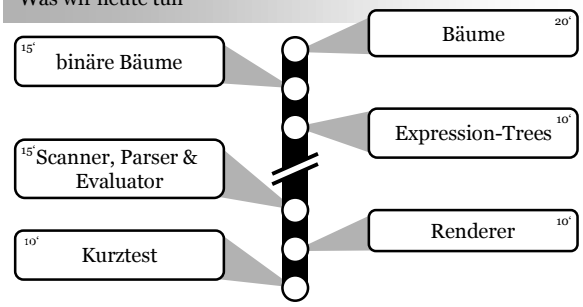
Informatik I WS 02/03

28. Januar 2003

- Mein Skript lesen, z.B. zum Thema *Bäume*
- Vordiplomsammlungen beim VIS

Was wir heute tun



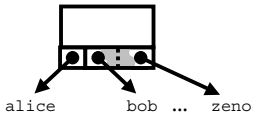
Bäume

wir haben lineare & doppelt verkettete Listen gesehen

Probleme / Nachteile von Listen?

Listenelement
hat 1 direkten
Nachfolger

Knoten (Baumelement)
hat mehrere direkte
Nachfolger

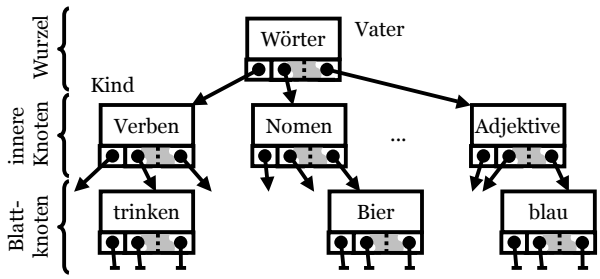


was ist daran jetzt besser?
wo brauchen wir das?

Baumstruktur

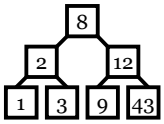
Baum gibt zusätzliche Struktur:

- die Nachfolger eines Knoten bilden eine Menge



Baumtiefe

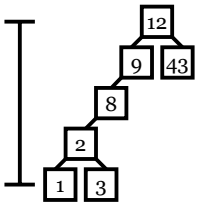
wie tief liegt ein Blatt von der Wurzel entfernt?



$tiefe_{min} = \lceil log_2(n) \rceil$

Was ist aber die Tiefe, wenn der Baum B nicht ausgeglichen ist?

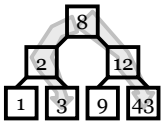
$$tiefe_B = \max_{k \in B} |weg(k)|$$



welcher Baum mit n Knoten hat die Tiefe n?
optimale Bäume haben minimale Tiefe

lineare Reihenfolge

keine eindeutige lineare Reihenfolge



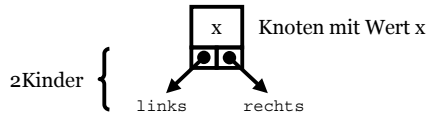
mehrere Wege wie man da drinnen herumirrt

Definition dreier Reihenfolgen:

- Preorder:
  eigener Wert, links, rechts 8 2 1 3 12 9 43
- Inorder:
  links, eigener Wert, rechts 1 2 3 8 9 12 43
- Postorder:
  links, rechts, eigener Wert 1 3 2 9 43 12 8

binäre Suchbäume

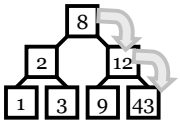
Bäume erleichtern die Suche erheblich



alle Knoten mit Wert < x alle Knoten mit Wert ≥ x

Beispiel Liste versus Baum:

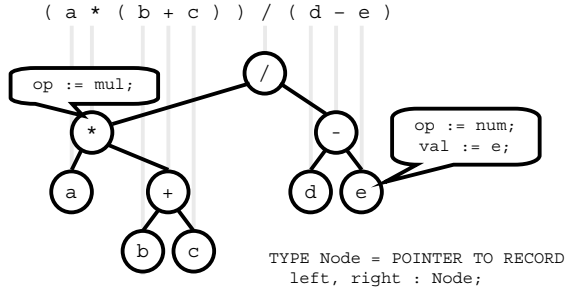
max. **4** Vergleiche



max. **2** Vergleiche

Expression-Tree

interne Datenstruktur für Auswertung von Ausdrücken




```
TYPE Node = POINTER TO RECORD
  left, right : Node;
  op : INTEGER;
  val : REAL;
END;
```

Evaluator – Auswerter des Expression-Tree

Monadisch = 1 Operator & **1** Operand: #A
Dyadisch = 1 Operator & **2** Operanden: A + B

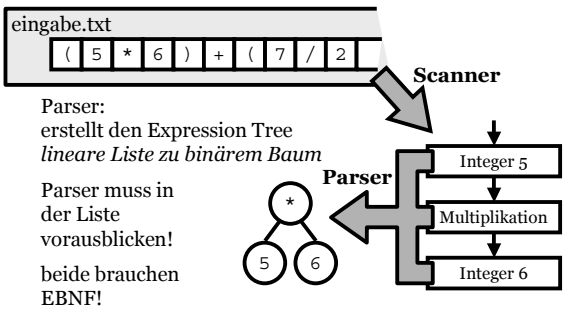
Auswertung mit Evaluator

```
PROCEDURE Val( x : Node ) : REAL ;
BEGIN
 IF x.right = NIL THEN              (* monadisch? *)
  IF x.left = NIL THEN              (* ein Blatt? *)
   RETURN x;          (* dann ist es das Resultat *)
  ELSE                          (* also monadisch *)
   RETURN Monadisch( x.op, Val(x.left) );
 END;
 ELSE                             (* also dyadisch *)
  RETURN Dyadisch( x.op, Val(x.left), Val(x.right));
 END
END Val;
```

aber wie kommt man zum Expression-Tree?

Scanner & Parser – Bauarbeiter des Expression-Tree

Scanner:
liest ein und erstellt erste Datenstruktur
lineare Bytefluss zu linearer Liste



Parser:
erstellt den Expression Tree
lineare Liste zu binärem Baum

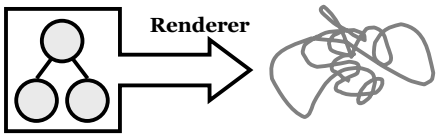
Parser muss in der Liste vorausblicken!

beide brauchen EBNF!

Renderer

to render – übertragen

Renderer überträgt von interner Datenstruktur in eine Grafik, die visuell darstellbar ist
interne Datenstruktur zu externer visualisierbarer Datenstruktur



Aufgabe Graphiksprache & Renderer

```
Graphic := { Declaration } Declaration.
Declaration := Name "=" Program.
Program := { Command }.
Command := "go" Dist
         | "line" Dist
         | "turn" Angle
         | "color" Colorcode
         | "call" Name Scale.
Dist := Real.
Angle := Real.
Scale := Real.
ColorCode := Integer.
Name := String.
```

```
DEFINITION SimpleGraphics;
  PROCEDURE Go(dist: REAL);
  PROCEDURE Line(dist: REAL);
  PROCEDURE Turn(angle: REAL);
  PROCEDURE Color(color: INTEGER)
END SimpleGraphics.
```

Was ist EBNF?
Wie internalisieren?
Was ist eine geeignete interne Datenstruktur?

Nächste Woche, 4. Februar 2002

30 Minuten Übungstunde mit folgenden Themen:

- Hash-Wert von Worten
- O-Notation
- Besprechung Ü10/Ü11

danach BQM Besuch