ETH zürich

Informatik II - Übung 03

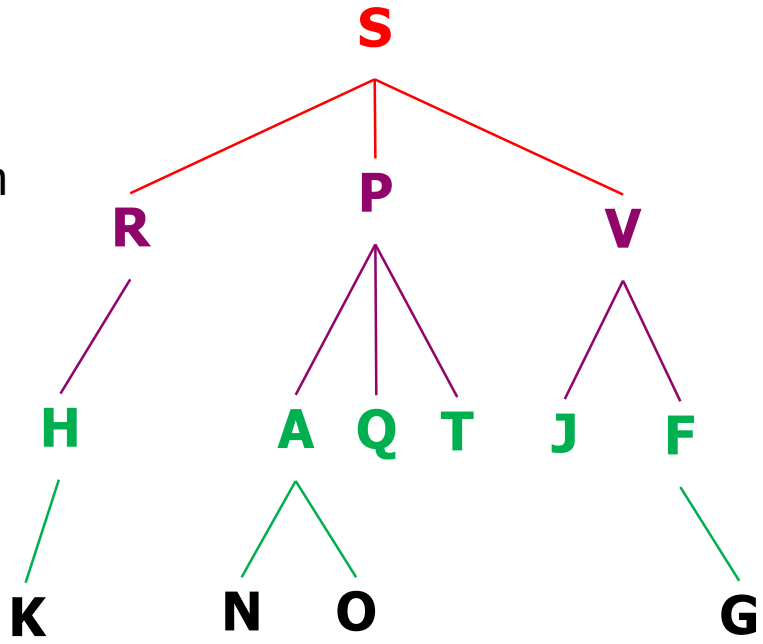
Katja Wolff
katja.wolff@inf.ethz.ch
15.03.2014

Besprechung Übungsblatt 2

U2.A1

- Klammerdarstellung und eingerückter Form

  S(R(H(**K**)),P(A(**N**,**O**),**Q**,**T**),V(**J**,F(**G**)))

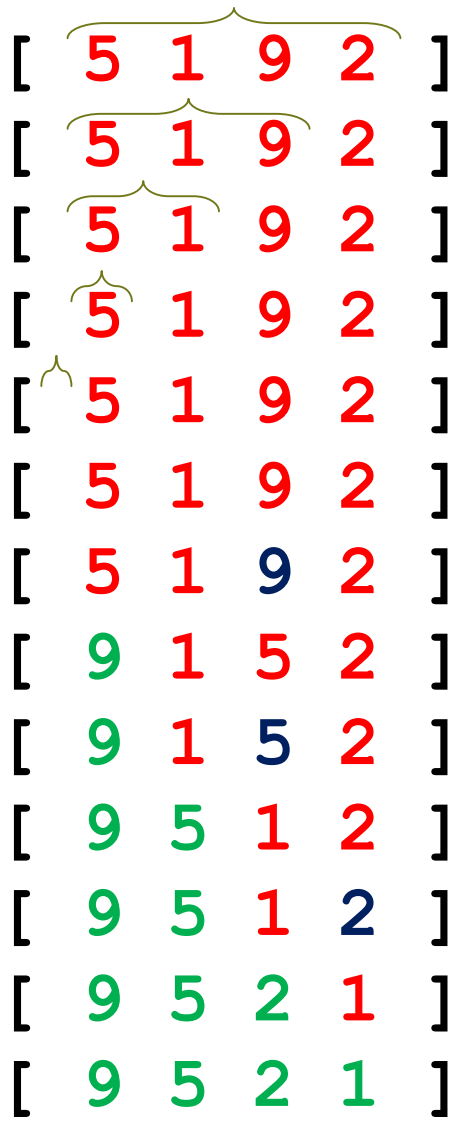


- Können Sie aus den obigen Links-klammerdarstellungen den dazugehörigen Baum eindeutig rekonstruieren?

→ **Ja,** falls Position der Knoten unwichtig (linker oder rechter Nachfolger?) - uneindeutig für H,K,G

→ **Falls Position der Knoten wichtig:** nutze "-" : z.B. H(K,-)

- Höhe? Längste Pfade? **B**lätter?




```
recursiveSort(4)
    recursiveSort(3)
        recursiveSort(2)
            recursiveSort(1)
                recursiveSort(0)
                Ist sortiert!
            9 <- findLargest(0,3)
            Swap
        5 <- findLargest(1,3)
        Swap
    2 <- findLargest(2,3)
    Swap
Kein swap mehr noetig...
```

→ Liste absteigend sortiert!

U2.A2

- Hilfsfunktion SWAP

```
/**
* swaps two fields of {@link RandomArray#numbers}
*
* @param i a valid index into {@link RandomArray#numbers}
* @param j a valid index into {@link RandomArray#numbers}
*/
private void swap(int i, int j)
{
     int tmp = numbers[j];
     numbers[j] = numbers[i];
     numbers[i] = tmp;
}
```

U2.A2

- SWAP innerhalb einer Schleife – ist das eine gute Idee?

```
void recursiveSort( int until ) {
    // 0 elements are considered to be sorted
    if( until == 0 )
       return;

    // sort first until-1 elements in the array
    recursiveSort( until - 1 );

    // bring the greatest element from the rest to position until-1
    for( int i = until; i < a.length; i++ )
    {
        if( a[i] > a[until-1] ){
          swap(until-1, i);
        }
    }
 }
```

U2.A2

- Besser: zuerst suchen, dann tauschen!

```
void recursiveSort( int until ) {
     // 0 elements are considered to be sorted
     if( until == 0 )
       return;

     // sort first until-1 elements in the array
     recursiveSort( until - 1 );

     // find index of greatest element after until-1
     int maxIndex = until - 1;
     for( int i = until; i < a.length; i++ ) {
        if( a[i] > a[maxIndex] ) {
           maxIndex = i;
        }
     }

     // swap elements at maxIndex and until-1
     swap( until-1, maxIndex );
}
```

bei 15 Werten durchschnittlich ~57 Swaps im ersten Fall und ~11 Swaps im zweiten Fall

U2.A2 – Coding style: No hardcoding

```
x < 10
```

```
x < a.length
```

```
if(myString.compareTo( ”hello world” ) == 0);
```

```
private static final String REF = ”hello world”;
…
if(myString.compareTo( REF ) == 0);
```

U2.A2 – Coding Style: Unterschied?

```
if (index >= boundary)
   return;
else if (array[index] == 'x')
   return;
```

```
if ( index >= boundary ||
     array[index] == 'x' )
   return;
```

Y im Ausdruck (X || Y) wird nur ausgewertet, falls X == false.

```
if (index < boundary)
   if (array[index] == 'x')
      array[index] = '\0';
```

```
if ( index < boundary &&
     array[index] == 'x' )
   array[index] = '\0';
```

Y im Ausdruck (X && Y) wird nur ausgewertet, falls X == true.

```
int counter = 0;
while (counter < n) {
   ...
   counter++;
}
```

Achtung: counter ist nach dieser Schleife immer noch definiert!

```
for ( int counter = 0;
      counter < n;
      counter++) {
   ...
}
```

Saubere Aufzählung. counter kann nach dem `for` erneut definiert werden.

U2.A2 – Coding style: Effizienz

Objektinstanzierung ist teuer!

```
void initialize() {
  for (int i=0; i<a.length; i++) {
     Random r = new Random();
     a[i] = r.nextInt(1000);
  }
}
```

```
void initialize() {
  Random r = new Random();
  for (int i=0; i<a.length; i++) {
     a[i] = r.nextInt(1000);
  }
}
```

U2.A3

- Wurzel an Index 0
- Direkte Nachfolger von i an (2i + 1) und (2i + 2)
- $2^{\text{Höhe-1}} = 2^{\text{Tiefe}} \leq \text{array.length} < 2^{\text{Tiefe+1}} = 2^{\text{Höhe}}$

```
int leftChild( node ){
        return 2 * node + 1;
}

int rightChild( node ){
        return 2 * node + 2;
}

int father( node ){
        return (node - 1) / 2;
}
```

(`father(0)` = -1 / 2 = 0)

U2.A3 b,c

- `toString()`
  - Rekursive Implementierung ist einfacher
- `checkTree()`
  - Teste jedes nichtleere Element auf einen nichtleeren Vater ("The root is its own father.”)
  - Teste auf nichtleeres Array

Ausblick: Übungsblatt 3

Hinweise U3

1. Programmverifikation und Schleifeninvarianten

2. Objekte und Referenzen (am Beispiel Strings)

3. Syntaxdiagramme
   - Diagramme gegeben; Welche Ausdrücke sind erzeugbar?

4. Syntaxchecker für Bäume
   - Ergänzen des Syntaxdiagramms aus der Vorlesung
   - Implementierung des Syntaxcheckers

U3.A1: Programmverifikation

```
static int f(int i, int j) {
   int u = i;
   int z = 0;

   while (u > 0) {
      z = z + j;
      u = u - 1;
   }
   return z;
}
```

a) Schleifeninvariante?

b) Korrektheitsbeweis?

c) Dummy-Statements:

```
z = z;
u = u;
```

U3.A2: String und StringBuffer

- Klasse `String` (immutable)
  - Operationen können gut optimiert werden
  - Aber: Modifikationen nur durch Kopie!

- Klasse `StringBuffer` (mutable)
  - Veränderbar ohne Kopie
  - Aber: Gewisse Operationen sind teuer!

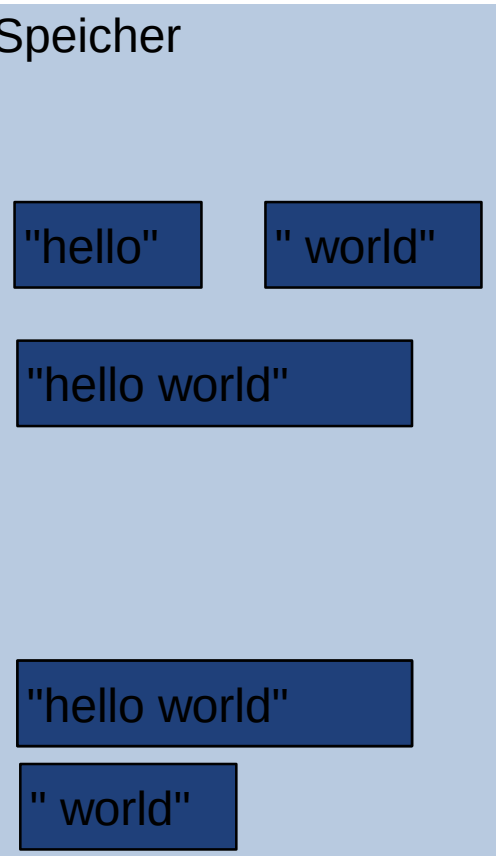
U3.A2 – String vs. StringBuffer

```
String myString = "hello";
myString = myString + " world";
```

String-Konkatenation in Java

```
StringBuffer myStringBuffer = "hello";
myStringBuffer.append(" world");
```

Methode von StringBuffer

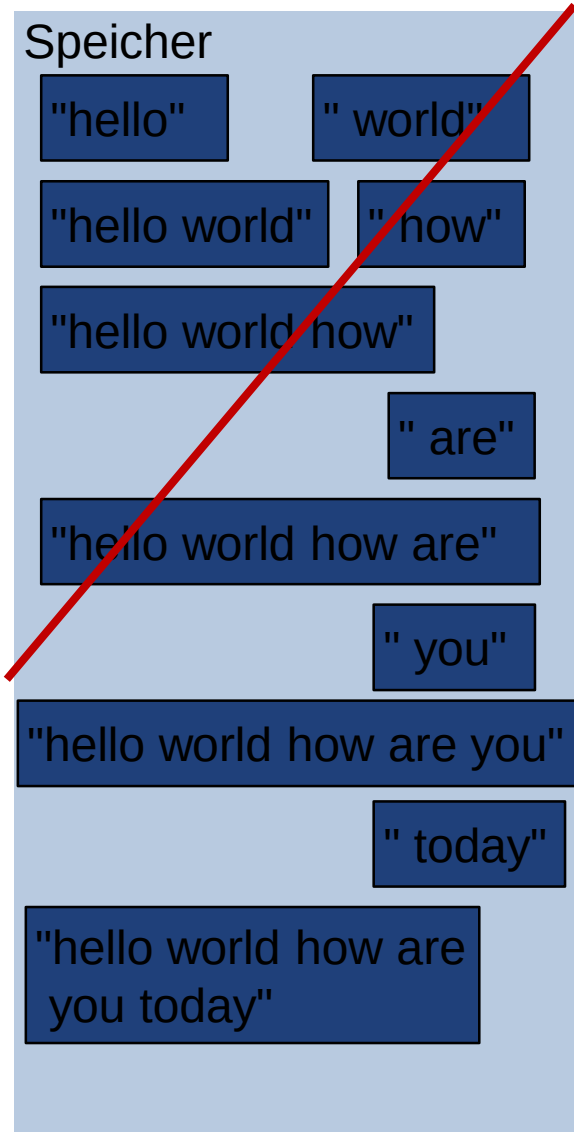


U3.A2 – String vs. StringBuffer

- Garbage Collector

```
String myString = "hello";
myString = myString+" world";
myString = myString+" how";
myString = myString+" are";
myString = myString+" you";
myString = myString+" today";
```

Garbage Collector



U3.A2 – Einen Blick wert: Eclipse-Debugger

- Vorsicht: Debugger zeigt temporäre Strings nicht an, da sie während eines Schrittes erzeugt und gleich wieder 'vernichtet' werden

```
String myString = "hello";
myString = myString+" world";
...
```

myString = "hello world"

…

U3.A2 - Programmanalyse

Welche Objekte existieren an den markierten Stellen zur Laufzeit?

- **Beispiel**: Referenzen und Objekte

```
String str = "foo";
StringBuffer buf = new StringBuffer("foobuffer");
```

  - Welche Objekte und Referenzen gibt es nun?

Objekte:

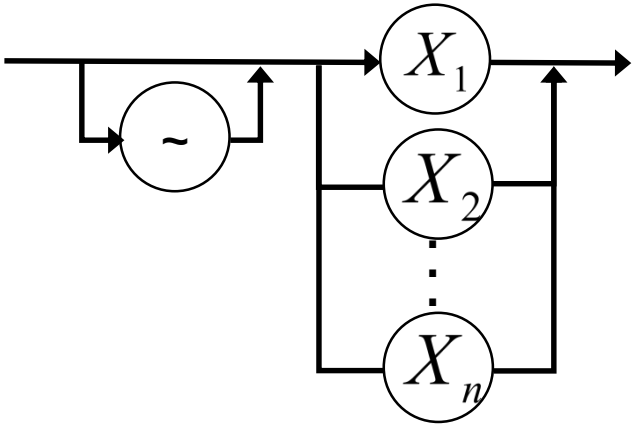
```
(1, String, "foo")
(2, String, "foobuffer")
(3, Stringbuffer, "foobuffer")
```

Referenzen:

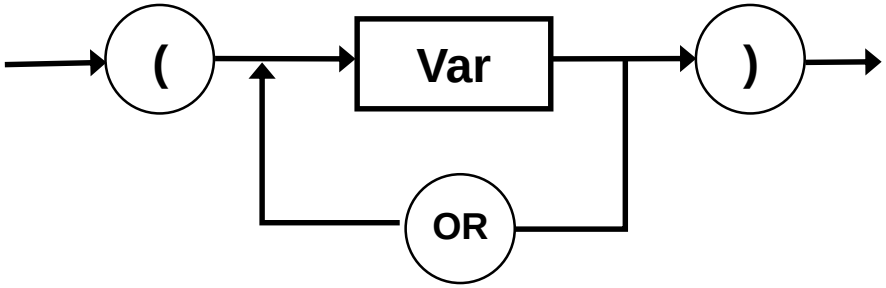
```
(str, 1)
(buf, 3)
```

U3.A3 - Syntaxdiagramme

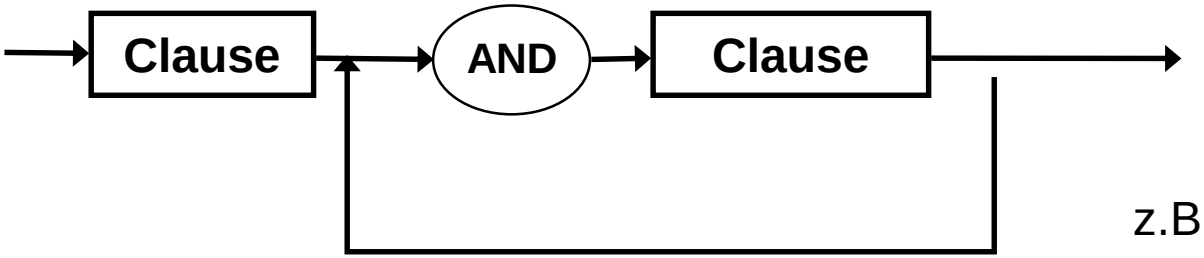
Var:



Clause:



Expr:



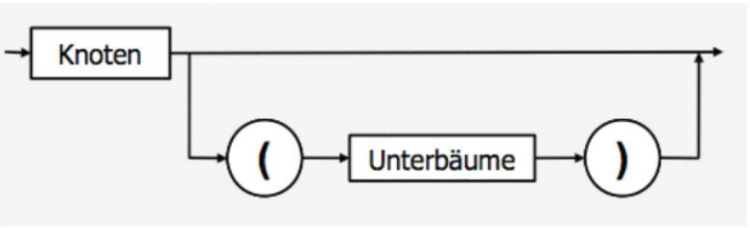
z.B $(\sim X_1 OR X_2) AND (X_n)$

U3.A4 – Syntax-Checker

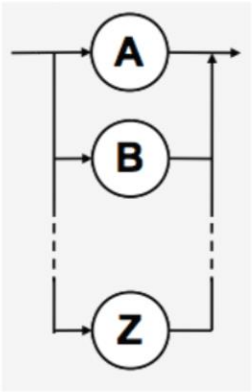
Ein Syntaxdiagramm-Beispiel: Klammerdarstellung eines Baums

Beispiel: A (B (D) , C (E , F))

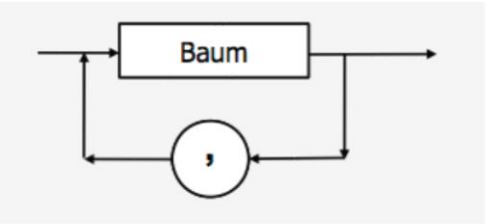
Baum:



Knoten:



Unterbäume :



Wie könnte man Binärbäume durch ein Syntaxdiagramm darstellen?

181

U3.A4 – Syntaxchecker für Bäume

Implementierung eines Syntaxcheckers für Bäume

- Zuerst: Erweiterung der Baumsyntax auf leere Teilbäume
- Syntaxchecker: Je eigene Methode für *Baum*, *Nachfolger* und *Knoten*

Hinweis: Lösung von 3a) muss mit eingebaut werden

Lösungsansatz: `public static int f(String str, int offset){...}`

Gebt jeweils den neuen **Offset** an die übergeordnete Funktion zurück. Überlegt euch, wie gross der Offset am Schluss sein muss und was passiert ist, falls er nicht genau diese Grösse hat.

Mögliche Probleme und dazugehörige Lösungsansätze/Erklärungen:

`StringIndexOutOfBoundsException` – ihr versucht auf den character `n` des Strings zuzugreifen, der Array ist aber weniger lang als `n`

…viel Spass!