Informatik II

Übungsstunde 3

simon.mayer@inf.ethz.ch
Distributed Systems Group, ETH Zürich

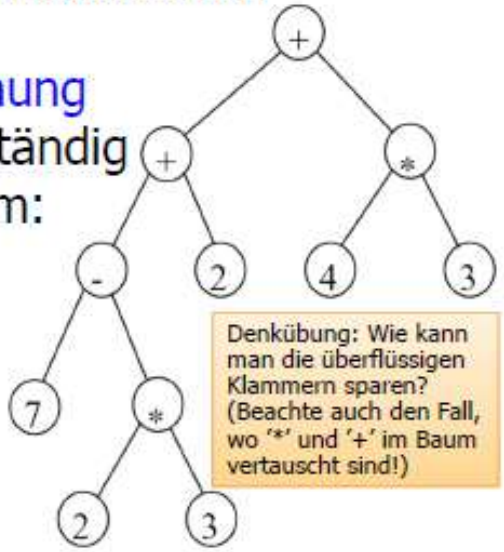
Ablauf

- Besprechung der Vorlesung
- Uebungsbezogene Themen:
  Strings, Syntaxdiagramme, Parser
- Zeit zum Programmieren...
  und fuer noch mehr Fragen

Symmetrisches Traversieren eines Binärbaums („inorder")

- Prinzip: Zuerst den linken Unterbaum traversieren, dann den Wert der Wurzel ausgeben, anschliessend den rechten Unterbaum traversieren

- Beispielhafte Anwendung: Rückgewinnung des Ausdrucks in Infix-Notation („vollständig geklammert") aus einem Operatorbaum:
  - Falls die Wurzel kein Blatt ist:
    - Ausgabe von '(' und wende Algorithmus auf linken Unterbaum an
  - Ausgabe des Attributs der Wurzel
  - Falls die Wurzel kein Blatt ist:
    - Wende Algorithmus auf rechten Unterbaum an und Ausgabe von ')'

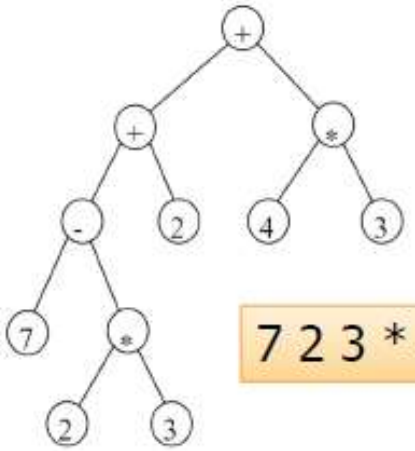


Denkübung: Wie kann man die überflüssigen Klammern sparen? (Beachte auch den Fall, wo '*' und '+' im Baum vertauscht sind!)

→ (((7-(2*3))+2)+(4*3))

Baumtraversierung in „postorder“

- Zuerst linken Unterbaum traversieren (falls vorhanden)
- Dann rechten Unterbaum traversieren (falls vorhanden)
- Dann die Wurzel „betrachten“
  - Z.B. das bei einem Operatorbaum vorhandene Attribut ausgeben



Lässt sich daraus der Operatorbaum wieder eindeutig rekonstruieren?

7 2 3 * - 2 + 4 3 * +

Das ist offenbar der Infix-Ausdruck ((7-(2*3))+2)+(4*3) in Postfix-Notation!

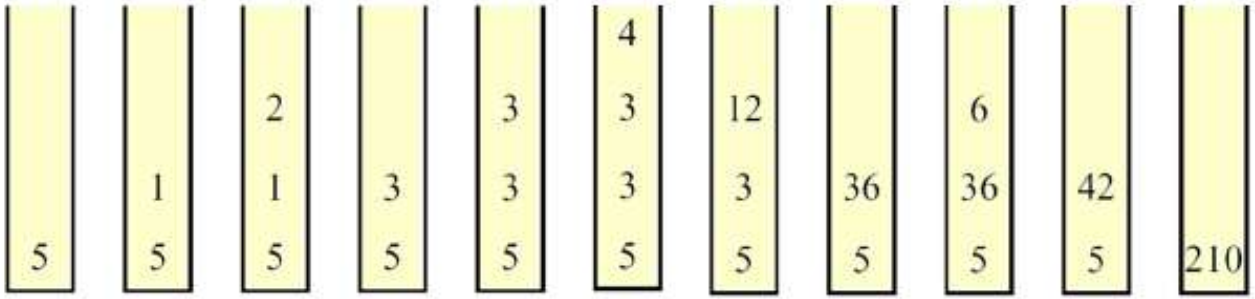
Postfix-Umwandlung mittels Stack

- Wir beschränken uns hier auf die beiden Operatoren + und * sowie einziffrige Operanden

- Lösungsidee:
  - Operanden (d.h. Zahlen) werden sofort ausgegeben
  - Operatoren kommen in den Stack
  - bei jeder schliessenden Klammer „pop": auf den obersten Operator anwenden (d.h. aus dem Stack holen) und ausgegeben
  - offenbar spielen öffnende Klammern keine Rolle, daher '(' genauso wie Leerzeichen etc. einfach überlesen!

Stackauswertung von 5 1 2 + 3 4 * * 6 + *

5 1 2 + 3 4 * * 6 + *



Ablauf

- Besprechung der Vorlesung
- Uebungsbezogene Themen:
  Strings, Syntaxdiagramme, Parser
- Zeit zum Programmieren...
  und fuer noch mehr Fragen

Übung 3

- 1. Objekte und Referenzen (am Beispiel Strings)
  - Programmausgabe
  - Programmanalyse: Objekte und Referenzen zur Laufzeit?
- 2. Syntaxdiagramme
  - Diagramme gegeben; Welche Ausdruecke sind erzeugbar?
- 3. Syntaxchecker fuer Baeume
  - Ergaenzen des Syntaxdiagramms aus der Vorlesung
  - Implementierung des Syntaxcheckers

Übung 3 – Aufgabe 1

- `String` – Immutable
  - Optimierungen moeglich weil statisch
  - Modifikationen nur durch Kopie

- `StringBuffer` – Mutable
  - Leicht veraenderbar (ohne Kopie)
  - Manche Operationen teuer (z.B. suchen)

- StringDemo

Übung 3 – Aufgabe 1

- Analyse des Programms

- Welche Objekte und Referenzen existieren an den markierten Stellen zur Laufzeit?

Übung 3 – Aufgabe 1

- Referenzen und Objekte – Beispiel

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

- Welche Objekte und Referenzen bestehen zur Laufzeit nach Ausfuehrung dieser Programmzeilen?
  - Objekte (plus IDs)

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

  - Referenzen (plus referenzierte Objekt-IDs)

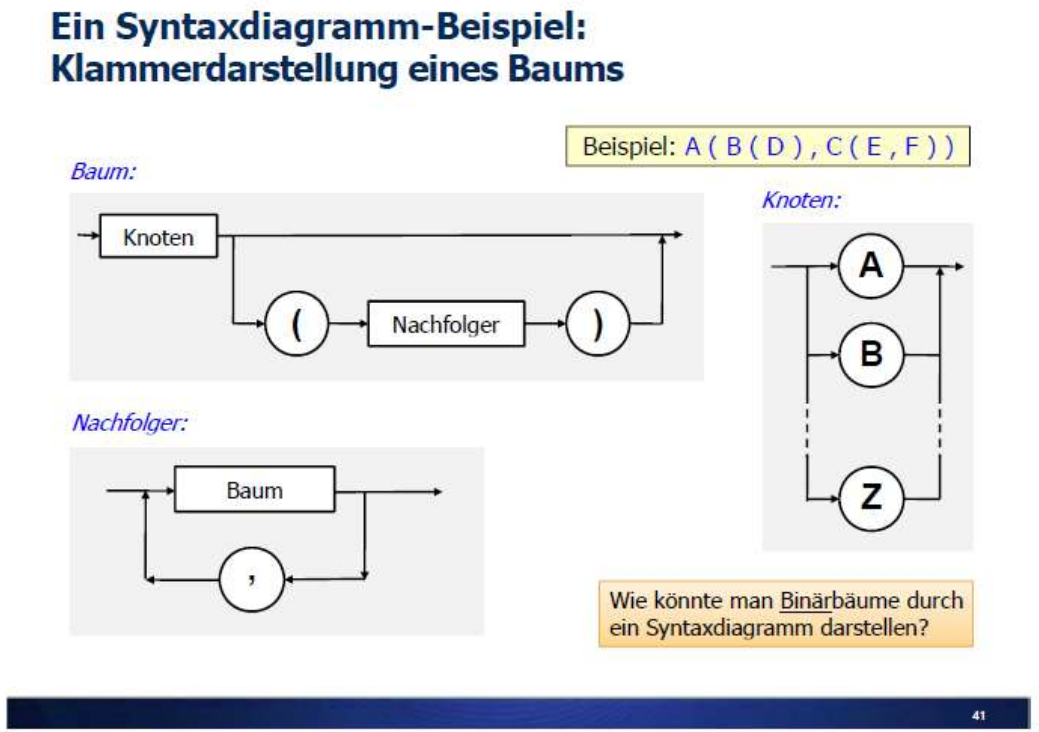
```
(str, 1)
(buf, 3)
```

Übung 3 – Aufgabe 2

- Syntaxdiagramme... wie in der Vorlesung vorgestellt
- Einfach!

Übung 3 – Aufgabe 3

- Erweiterung der Baumsyntax auf leere Teilbaeume



Übung 3 – Aufgabe 3

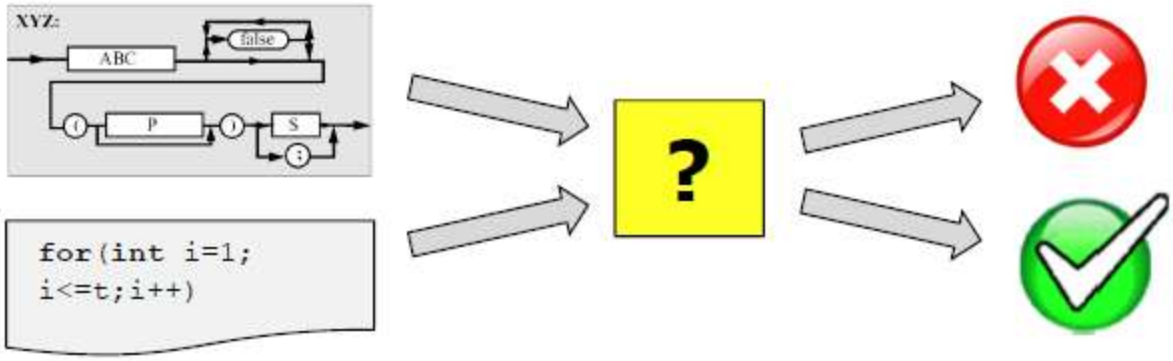
- Rekursiver Syntaxchecker
  - Je eigene Methode fuer «Baum», «Nachfolger» und «Knoten»

Syntaxanalyse durch „rekursiven Abstieg"

- Problem: Einen Algorithmus angegeben, der für
  - ein beliebiges Syntaxdiagramm und
  - einen beliebigen Text

  eindeutig entscheidet, ob sich der Text mit dem Diagramm generieren lässt (→ Syntaxchecker)

  → Theorie der Syntaxanalyse → formale Sprachen, Compilerbau



Ein Parser als Java-Programm (2)

```
//Anfangsteil kommt später
  ...
  void int_const(){...}
  void Ausdruck(){...}
  void Term() {
    Faktor();
    while (c == '*') {
      c = KdbInput.getc();
      Faktor();
    }
  }
  void Faktor() {
    if ((c >='0') && (c <='9'))
      int_const();
    else
      if (c == '(') {
        c = KdbInput.getc();
        Ausdruck();
        if (c == ')') c = KdbInput.getc();
        else Fehler(1);
        }
      else Fehler(2);
    }
  }
```

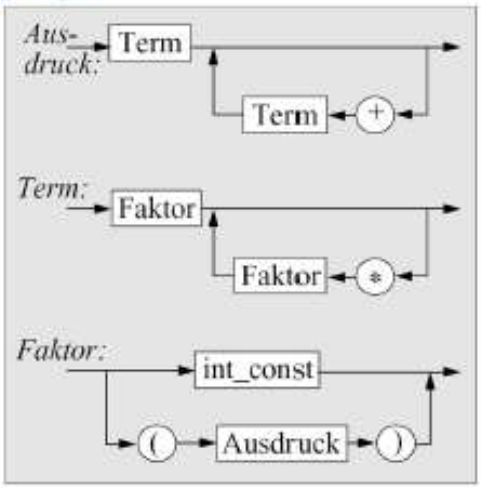
"Term an Faktor: 'Lies einen Faktor!'"

"Faktor an int_const: 'Lies eine int_const!'"

Rekursion!

```
public static void main(String args[]) {
  Parser p = new Parser ();
  p.c = KdbInput.getc();
  p.Ausdruck;
}
```

"Hauptprogramm an Parser p: 'Lies einen Ausdruck!'"



Ablauf

- Besprechung der Vorlesung
- Uebungsbezogene Themen:
  Strings, Syntaxdiagramme, Parser
- Zeit zum Programmieren...
  und fuer noch mehr Fragen

Projekt: Einfacher Syntaxchecker

Informatik II

Übungsstunde 3

simon.mayer@inf.ethz.ch
Distributed Systems Group, ETH Zürich