TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Software Entwicklung 1

Annette Bieniusa / Arnd Poetzsch-Heffter

AG Softech
FB Informatik
TU Kaiserslautern

Rekursion

Motivation

- Rekursion ist eine elegante Strategie zur Problemlösung, die es erlaubt eine Problemstellung auf eine einfachere Problemstellung der selben Art zurückzuführen.
- *Idee:* Prozeduren können nicht nur beliebige (andere) Prozeduren während ihrer Ausführung aufrufen, sondern auch sich selbst.

Beispiel: Fibonacci-Folge I

Die Fibonacci-Folge tritt bei diversen Naturphänomen in Erscheinung.

0 1 1 2 3 5 8 13 21 34 55 89 144 233 ...

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

Beispiel: Fibonacci-Folge II

```
public static int fib(int n) {
  if (n == 0) {
    return 0;
  }
  if (n == 1) {
    return 1;
  }
  return fib(n-2) + fib(n-1);
}
```

- Der **Basisfall** liefert einen Wert ohne einen nachfolgenden rekursiven Aufruf.
- Der **Reduktionsschritt** verknüpft den Prozeduraufruf mit dem Prozeduraufruf für (andere) Parameterwerte.
- *Wichtig:* Der Basisfall muss nach endlich vielen rekursiven Aufrufen erreicht werden, damit das Programm terminiert.
  Dies ist in dieser Implementierung nur für Parameter $n \geq 0$ der Fall.

Definition: Rekursive Prozedurdeklaration

Eine Prozedurdeklaration P heißt **direkt rekursiv**, wenn der Prozedurrumpf einen Aufruf von P enthält.

Eine Menge von Prozedurdeklarationen heißen **verschränkt rekursiv** oder **indirekt rekursiv** (engl. *mutually recursive*), wenn die Deklarationen gegenseitig voneinander abhängen.

Eine Prozedurdeklaration heißt **rekursiv**, wenn sie direkt rekursiv ist oder Element einer Menge verschränkt rekursiver Prozeduren ist.

Beispiel: Verschränkte Rekursion

```
public static boolean istGerade(int n) {
  if (n == 0) {
    return true;
  } else {
    return istUngerade(n-1);
  }
}

public static boolean istUngerade(int n){
  if (n == 0) {
    return false;
  } else {
    return istGerade(n-1);
  }
}
```

Globale und lokale Variablen

S

Globale und lokale Variablen I

Jede Variablendeklaration definiert eine **Programmvariable**.

Wir unterscheiden hier:

- **globale Variablen**: Variablen, deren Deklaration ausserhalb von Prozeduren erfolgt
- **(prozedur-)lokale Variablen**: Variablen, deren Deklaration innerhalb einer Prozedur steht

Globale und lokale Variablen II

Jeder Programmvariablen entsprechen zur Ausführungszeit/Laufzeit des Programms eine oder mehrere Speichervariablen:

- Jeder globalen Variablen ist genau eine Speichervariable zugeordnet.
- Ist v eine lokale Variable zur Prozedur p, dann gibt es zu jeder Inkarnation von p eine Speichervariable für v.
- (Andere Variablentypen behandeln wir später.)

Begriffsklärung: Lebensdauer von Variablen

Die **Lebensdauer** einer Speichervariable ist der Teil des Ablaufs, in dem sie für die Ausführung bereit steht.

Die Lebensdauer globaler Variablen erstreckt sich über den gesamten Ablauf des Programms.

Variablen zu lokalen Deklarationen leben solange wie die zugehörige Prozedurinkarnation bzw. über die Dauer der Ausführung ihres Blocks.

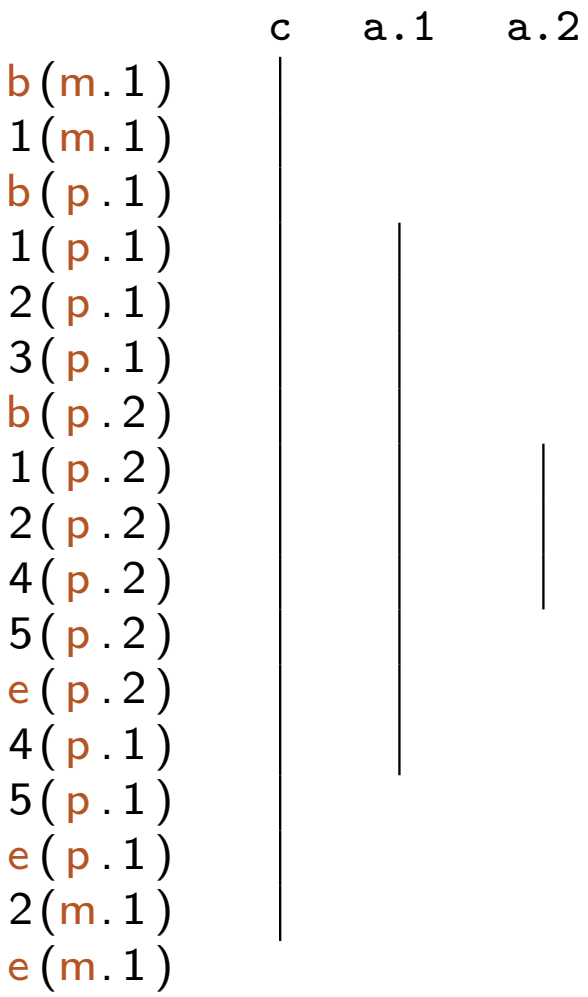
Beispiel: Lebensdauer

```
public class LebensdauerTest {
  public static int c = 5;

  public static void p(int i) { //b(p)
    int a;
    a = i;                      //1(p)
    if (a == 2) {               //2(p)
      p(a-1);                   //3(p)
    }
    StdOut.println(a);          //4(p)
    StdOut.println(c);          //5(p)
  }                             //e(p)

  public static void main(String[] args){  //b(m)
    p(2);                                  //1(m)
    StdOut.println(c);                     //2(m)
  }                                        //e(m)
```

Wir betrachten den Ablauf des obigen Programms mit den Markierungen und die Variablenlebensdauer:



Lebensdauer von Feldern

Die Lebensdauer eines Felds

- beginnt mit der Erzeugung und
- endet mit der Terminierung des Programms.

Beispiel:

```
static int[] erzeugeFeld (int n) {
  if (n < 0) {
   n = 0;
  }
  return new int[size];
}

static void verwendeFeld() {
  int[] a = erzeugeFeld(78);
  ...
}
```

Beispiel: Indentierung / Einrückung I

```
// globale Variablen
public static final int TAB = 4;
public static int indent = 0;    // aktuelle Einruecktiefe

public static void drucke(String s) {
  for(int i = 0; i < indent; i++) {
    StdOut.print(" ");
  }
  StdOut.println(s);
}
```

Beispiel: Indentierung / Einrückung II

```
public static void s() {
  drucke("Betrete s");
  indent += TAB;
  indent -= TAB;
  drucke("Verlasse s");
}
```

```
public static void q (int iq) {
  drucke("Betrete q");
  indent += TAB;
  p(iq);
  indent -= TAB;
  drucke("Verlasse q");
}
```

```
public static void p (int ip) {
  drucke("Betrete p");
  indent += TAB;
  if (ip > 2) {
    p(2);
    q(2);
    r(2);
  }
  indent -= TAB;
  drucke("Verlasse p");
}
```

```
public static void r (int ir) {
  drucke("Betrete r");
  indent += TAB;
  if (ir < 2) {
    s();
  }
  indent -= TAB;
  drucke("Verlasse r");
}
```

Beispiel: Indentierung / Einrückung III

- Welche dieser Prozeduren sind rekursiv, direkt rekursiv, verschränkt rekursiv?

Beispiel: Indentierung / Einrückung IV

- `r` und `s` sind nicht rekursiv.
- `q` ist verschränkt rekursiv.
- `p` ist direkt und verschränkt rekursiv.

Bemerkungen

- `i += j;` ist eine Kurzschreibweise für die Zuweisung `i = i + j;` (analog: `-=`, `*=`).
- In Java können statischen Methoden und Variablen vor ihrer Deklaration verwendet werden. Daher ist die Reihenfolge der Deklaration beliebig.
- Variablen, die als `final` deklariert werden, können nach der Initialisierung nicht mehr verändert werden. Sie eignen sich um beispielsweise Konstanten wie `TAB` zu repräsentieren.

  Finale Referenzvariablen verweisen immer auf auf das Objekt, mit dessen Referenz die Variable initialisiert wurde. Diese Objekt kann aber weiterhin verändert werden!
- Die Verwendung von (veränderlichen) globalen Variablen kann das Testen von Methoden erschweren und führt bisweilen zu schwer verständlichem Code. Daher versuchen wir globale Variablen hier möglichst zu vermeiden.
- Anstelle der Deklaration der globalen Variablen `indent` kann man alternativ die Methoden um zusätzliche Parameter erweitern.

Begriffsklärung: statisch/dynamisch

Statisch bezeichnet man in der Programmiertechnik alle Aspekte, die sich auf das Programm beziehen und die man aus ihm ersehen kann, ohne es auszuführen. Statische Aspekte sind unabhängig von Eingaben.

Dynamisch bezeichnet man alle Aspekte, die sich auf die Ausführungszeit beziehen.

Beispiele: statisch/dynamisch

Statisch:

- Programmvariablen
- Prozedurdeklarationen
- Anweisungen
- Aspekte der Übersetzung

Dynamisch:

- Speichervariablen
- Prozedurinkarnationen
- Ablauf, Ausführung
- Aufrufbäume
- Lebensdauer (von Prozedurinkarnationen, usw.)

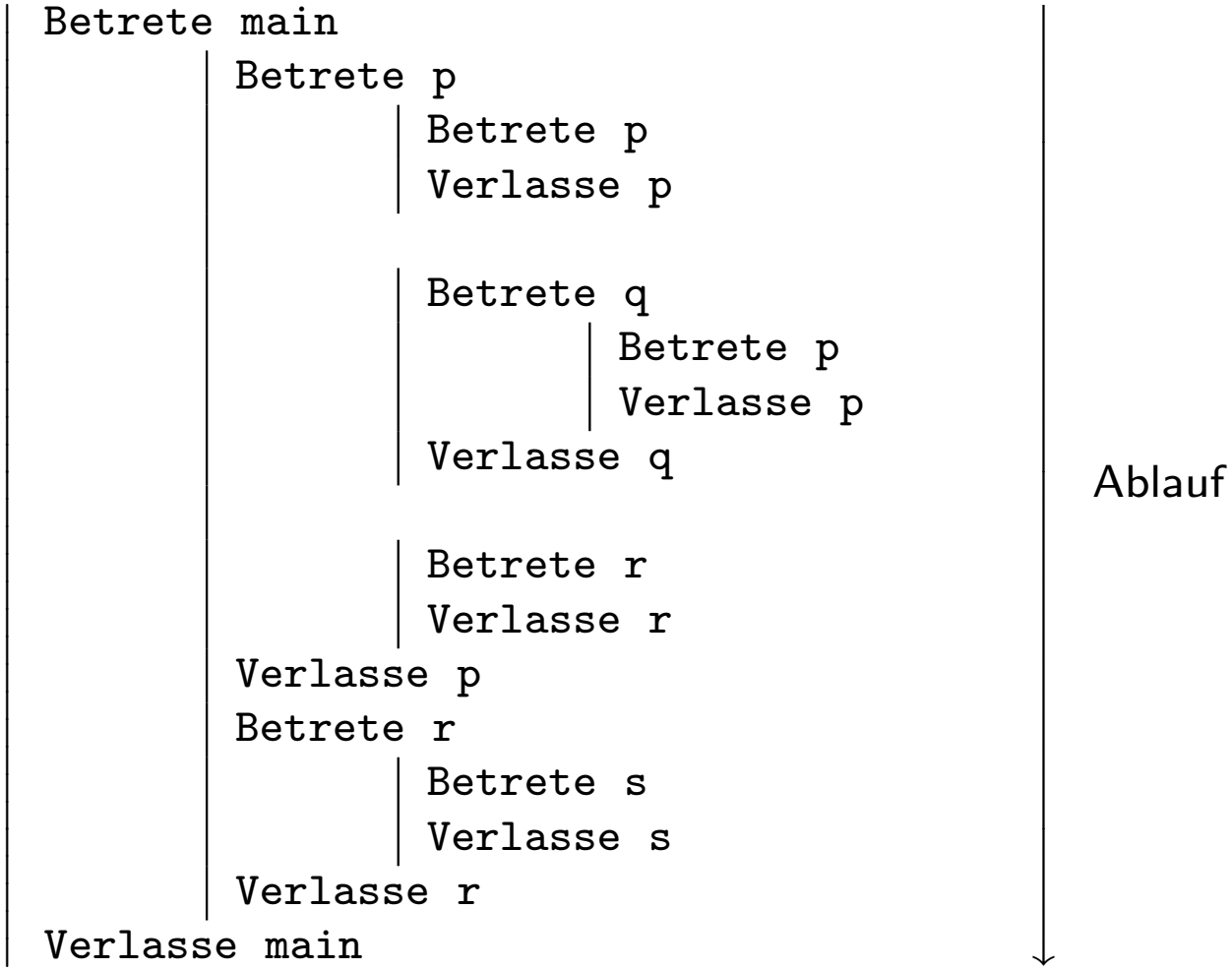
Dynamische Aspekte von Prozeduraufrufen I

Hauptprozedur:

```
public static void main( String[] args ) {
    drucke("Betrete main");
    indent += TAB;
    p(3);
    r(1);
    indent -= TAB;
    drucke("Verlasse main");
}
```

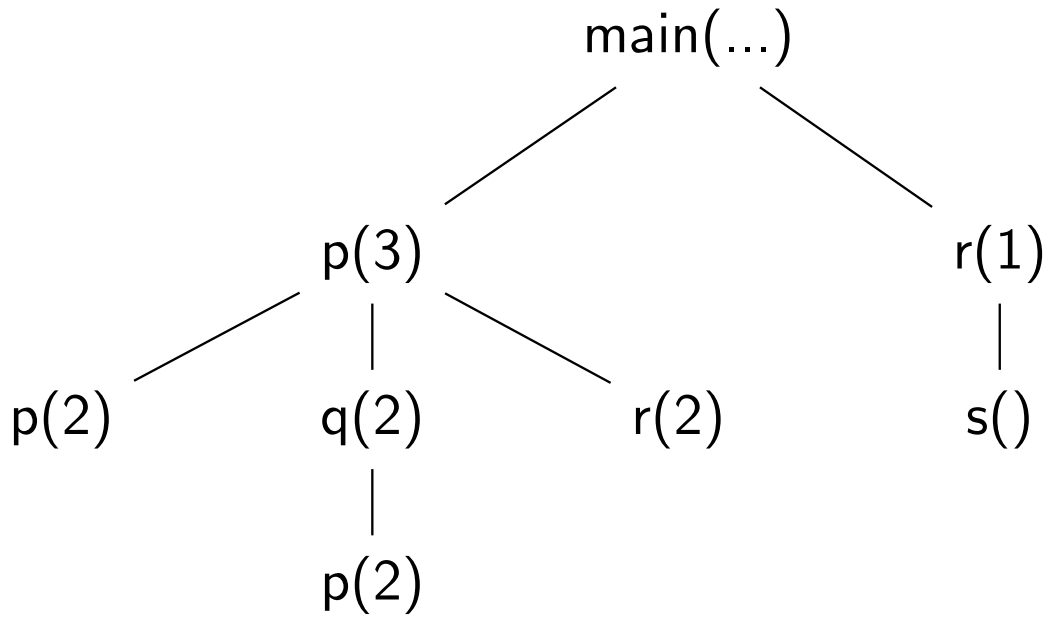
Jeder Balken entspricht einer Prozedurinkarnation. Es gibt mehrere Inkarnationen der gleichen Prozedur.

Dynamische Aspekte von Prozeduraufrufen II



Bemerkung

Eine wichtige Struktur des Ablaufs prozeduraler Programme ist der Aufrufbaum.
Beispiel:



Die Lebensdauern von Inkarnationen der gleichen Prozedur können sich überlappen (im Beispiel: p(3) und p(2)).

Begriffsklärung: Inkarnation / Lebensdauer

Beim Aufruf einer Prozedur p wird eine neue **Inkarnation** von p erzeugt.
Dabei wird:

- Speicherplatz für die lokalen Variablen angelegt;
- die Anweisung gemerkt, an der nach Ausführung der Prozedurinkarnation fortzusetzen ist.

Die **Lebensdauer** einer Prozedurinkarnation beginnt mit deren Erzeugung und endet mit der Ausführung des Rumpfes zu dieser Inkarnation.

Sie erstreckt sich über einen Teil des Programmablaufs.

Begriffsklärung: Prozeduraufrufbaum

Der **Prozeduraufrufbaum** beschreibt die Prozeduraufrufstruktur in einem Programm- oder Prozedurablauf.

Seine Baumknoten sind mit Prozedurinkarnationen markiert, so dass jede Inkarnation p_i genau die Inkarnationen als Kinder hat, die von p_i aus erzeugt wurden und zwar in der Reihenfolge des Ablaufs.

Die Lebensdauern von Inkarnationen der gleichen Prozedur können sich überlappen.

Deshalb muss es ggf. mehrere Kopien/Instanzen der gleichen lokalen Variablen geben.

Zusammenfassung

- Rekursion (direkte, verschränkte)
- Lokale und globale Variablen
- Lebensdauer von Variablen
- Prozerdurinkarnationen
- Prozeduraufrufbäume