Programmierstarthilfe SS 2009

Fakultät für Ingenieurwissenschaften und Informatik

5. Blatt

Für die Woche vom 25.5. bis zum 29.5.2009 (KW 22)

Organisatorisches

Die Webseiten zur Veranstaltung sind unter `http://www.uni-ulm.de/in/mi/lehre/2009ss/programmierstarthilfe.html` zu finden.

Diese Woche lernen wir mit Rekursion zu arbeiten. Am Beispiel der Fakultätsfunktion schauen wir uns an wie man ein Programm aufbaut und wie die Gültigkeitsbereiche bei der Rekursion funktionieren.

0.1 Fakultät rekursiv

Es soll eine Methode `fakultaet(int n)` geschrieben werden, die den Wert $n!$, also $1 \cdot 2 \cdot \ldots \cdot n$, berechnen und zurückgeben soll.

a) Überlege dir eine rekursive Lösung für das Problem. Mache dir zunächst den Basisfall für eine Abbruchbedingung klar. Wie muss nun der Rekursionsfall aussehen? Schreibe auf einem Blatt Papier auf, wie die Fakultät rekursiv entsteht. Was ist dir lieber: $1 \cdot 2 \cdot \ldots$, oder $n \cdot n-1 \cdot \ldots$?

b) Implementiere nun die rekursive Methode in Java und teste sie.

c) Implementiere zum Schluss die Methode iterativ, also mit einer Schleife.

1 Aufgaben

1.1 Fibonacci-Zahlen

Eine typische Standardaufgabe für Rekursion sind die Fibonacci-Zahlen, eine Reihe von Zahlen $f(i)$. Diese sind rekursiv definiert: $f(0) = 0$, $f(1) = 1$, $f(n) = f(n-1) + f(n-2)$

a) Schreibe ein rekursive Methode für die Berechnung des n-ten Reihenelements der Fibonacci-Reihe, das die obige Definition nutzt.

b) Da jedes Element die Summe seiner beiden Vorgänger ist, kann man die Berechnung auch recht gut iterativ vornehmen. Dafür beginnt man bei $f(0)$ und $f(1)$, addiert diese zu $f(2)$ und merkt sich $f(1)$ und $f(2)$ für den nächsten Schritt, dann addiert man diese usw. Schreibe diese iterative Methode.

c) Mit `System.currentTimeMillis()` erhälts du die aktuelle Systemzeit in Millisekunden als `long`. Miss damit die jeweilige Laufzeit beider Methoden für etwas größere n, z.B. $n = 45$ ($n > 46$ erzeugt bereits einen Wert außerhalb des Integer-Bereichs).

d) Wie kommt es zu den Laufzeitunterschieden? Mach dir z.B. klar, wie die rekursive Variante arbeitet, indem du einen Aufrufbaum zeichnest (siehe Kapitel Rekursion im Skript).

e) Rekursion scheint langsam bzw. ineffizient zu sein. Stimmt diese Aussage? Wann ist sie wahr und wann nicht? Gibt es Probleme, die man mit Rekursion elegant und zugleich schnell lösen kann?

1.2 Rekursiver Palindromtest

Es soll eine Methode `istPalindrom(String wort)` geschrieben werden, die den übergebenen String rekursiv auf Symmetrie testet und zurückgeben soll, ob es sich um ein Palindrom handelt oder nicht.

a) Mache dir zunächst Gedanken darüber, wie du bisher auf Symmetrie getestet hast und wie du rekursiv vorgehen könntest. Wichtig sind ein korrekter Basisfall sowie Rekursionsfälle. Wie lange musst du rekursiv absteigen? Wann kann abgebrochen werden? Wie transportierst du das Zwischenergebnis?

b) Implementiere die Methode nun in Java. Es gibt mehrere Lösungsansätze, für einen kann es nützlich sein, eine überladene Methode `istPalindrom(...)` zu schreiben, in der du mehr Parameter mitgeben kannst. Die Methode `istPalindrom(String wort)` ruft dann die überladene Methode mit Startwerten auf.

1.3 Rekursion nachvollziehen

Betrachte folgende rekursive Methode:

```
1 static int ackermann(int n, int m){
2         if(n == 0) return m + 1;
3         else if(m == 0) return ackermann(n - 1, 1);
4         else return ackermann(n -1, ackermann(n, m-1));
5 }
```

Was gibt sie für den Aufruf ackermann(2, 2) zurück? Vollziehe zur Beantwortung dieser Frage die Aufrufstruktur auf dem Papier nach. Keinen Computer verwenden!

2 Bonusaufgaben

Durch die Aufgaben oben hast du ein Verständnis für das neue Konzept dieses Blatts bekommen. Durch Bonusaufgaben kannst du nun deine Kenntnisse vertiefen.

2.1 Potenzierung rekursiv

Es soll eine Methode `potenziere(int a, int b)` geschrieben werden, die den Wert a^b berechnen und zurückgeben soll.

a) Implementiere zunächst die Methode iterativ, also mit einer Schleife.

b) Implementiere die Methode mit Hilfe der Methode `Math.pow(double, double)`. Schau hierfür in die API-Dokumentation, wie diese Methode verwendet wird.

c) Überlege nun eine rekursive Lösung für das Problem. Mache dir zunächst den Basisfall für eine Abbruchbedingung klar. Wie muss nun der Rekursionsfall aussehen? Skizziere den Ablauf der rekursiven Berechnung auf einem Blatt Pappier.

d) Implementiere nun die rekursive Methode in Java und teste sie.

e) Es gilt $a^{2n} = a^n \cdot a^n$. Kannst du damit deinen Algorithmus noch optimieren?

2.2 Primfaktorzerlegung

a) Schreibe ein Programm, welches eine benutzerspezifizierte natürliche Zahl rekursiv in ihre Primfaktoren zerlegt und diese ausgibt. Überlege dir dazu zunächst, wie man prüfen kann, ob eine Zahl durch eine andere teilbar ist. Wie sieht dann der Rekursionschritt aus? Was ist der Basisfall?

b) Vollziehe auf dem Papier nach, was bei der Primfaktorzerlegung von 42 passiert und überprüfe deine Überlegungen durch Ausgaben in deinem Programm.

2.3 Produkt rekursiv

Es soll eine Methode `produkt(int a, int b)` geschrieben werden, die das Produkt von a bis b ($\prod_{i=a}^{b} i$), berechnen und zurückgeben soll.

a) Überlege dir eine rekursive Lösung für das Problem. Mache dir zunächst den Basisfall für eine Abbruchbedingung klar. Wie muss nun der Rekursionsfall aussehen? Schreibe auf einem Blatt Papier auf, wie das Produkt rekursiv entsteht.

b) Implementiere nun die rekursive Methode in Java und teste sie.

c) Implementiere zum Schluss die Methode iterativ, also mit einer Schleife.

2.4 Summenbildung rekursiv

Es soll eine Methode `summiere(int n)` geschrieben werden, die den Wert $\sum_{i=1}^{n} i$ berechnen und zurückgeben soll.

a) Implementiere zunächst die Methode iterativ, also mit einer Schleife.

b) Überlege nun eine rekursive Lösung für das Problem. Mache dir zunächst den Basisfall für eine Abbruchbedingung klar. Wie muss nun der Rekursionsfall aussehen? Schreibe auf einem Blatt Papier auf, wie die Summe rekursiv entsteht. Gibt es mehrere Möglichkeiten, aufzusummieren?

c) Implementiere nun die rekursive Methode in Java und teste sie.

2.5 *Rekursiver Würfel

Nun soll eine Methode `wuerfeln(int anzahl)` programmiert werden, die alle möglichen Kombinationen bei einem Würfel mit sechs Seiten und einer gegebenen Anzahl von Würfeln erzeugen und ausgeben soll. Bei der Anzahl drei sollen zum Beispiel alle Ergebnisse 1-1-1, 1-1-2, ... 6-6-5, 6-6-6 ausgegeben werden.

a) Sicher kennst du aus der Schule noch Baumdiagramme für Wahrscheinlichkeiten. Erstelle auf Papier ein solches Diagramm für zweimaliges Würfeln. Ignoriere die Wahrscheinlichkeiten und beschrifte die Blätter mit der entstandenen Kombination.

b) Implementiere nun die Methode `wuerfeln(int anzahl)` rekursiv. Wie oft musst du pro Rekursionschritt verzweigen?

3 Für das nächste Blatt

Nächste Woche ist Reading Week. Wir werden mit euch ein kleines Projekt machen und neuen Stoff erst in der Woche danach behandeln.