Javakurs
SS 2006, Freitagsrunde und Tutoren

Aufgabenblatt 2

letzte Aktualisierung: 26. März, 12:39

Ausgabe: 03.04.2006

Thema: *Methoden und Schleifen*

Hinweis: Die Vorgaben zu den einzelnen Aufgaben findet ihr unter:

`~seclab/javakurs/le2`

1. Aufgabe: Sprachliche Konstrukte (arrays, while)

Analysiere die gegebenen Methoden einzeln und finde für jede heraus, wo das Problem liegt. Dafür kannst Du eigene Ausgaben einbauen. Um die Methoden einzeln testen zu können, kommentiere jeweils die Methodenaufrufe aus der `main`-Methode aus, die du nicht testen willst.

2. Aufgabe: Die Fibonacci-Funktion

$$fib(x) := \begin{cases} x & \text{falls } x < 2 \\ fib(x-1) + fib(x-2) & \text{sonst} \end{cases}$$

$$fib := 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ..$$

Opal-Code:

```
FUN fib : nat -> nat
DEF fib(n) ==
   IF n<2
   THEN n
   ELSE fib(n-1)+fib(n-2)
   FI
```

2.1. Simple Implementierung in Java Implementiere die Fibonacci-Funktion in JAVA. Orientiere Dich dabei möglichst stark am Opal–Code bzw. der mathematischen Definition.

2.2. Effizientere Implementierung in Java Ändere Deine Implementierung und passe diese an die folgende (effizientere) Opal–Implementierung an.

```
FUN fib: nat -> nat
DEF fib(x) ==
   IF x<2
   THEN x
   ELSE fibAcc(2, 1, 1, x)
   FI

FUN fibAcc: nat ** nat ** nat ** nat -> nat
-- idx : momentaner Index
-- p1  : fib(idx)
-- p2  : fib(idx-1)
-- max : der Index des zu berechnenden Folgengliedes
DEF fibAcc(idx, p1, p2, max) ==
   IF idx<max THEN fibAcc(idx+1, p1+p2, p1, max)
   IF idx=max THEN p1
   FI
```

2.3. Implementierung in Java Verwende anstatt einer Rekursion eine Iteration.

2.4. Mehrere Fibonacci-Zahlen in einem Array Der Anwender möchte nun die ersten n Fibonacci-Zahlen berechnen. Finde eine effiziente Implementierung mit folgender Signatur: `static int[] fib(int max)`

3. Aufgabe: Primzahlen

3.1. Primzahlprüfung Die Methode `isPrime` soll überprüfen ob die übergebene Zahl i eine Primzahl ist.

$$i\ Prim \Leftrightarrow i \in \mathbb{N} \wedge i > 1 \wedge \forall j \in \{2, .., i-1\} : i \not\equiv 0 \ mod\ j$$

Primzahlen sind also: $2, 3, 5, 7, 11, 13, 17, 19, 23, ..$

3.2. Methodenuntersuchung Untersuche die Methode `findPrimes`.

- Was tut sie (genau)?
- Bei welchen Eingaben tauchen Fehler auf?
- Wieso ist der Code nicht schön?
- Wo verliert die Funktion Zeit?
- Was könnte man verbessern?

Baue für Deine Untersuchung zusätzliche Ausgaben in der Funktion `findPrimes` ein und analysiere die Ausgaben der Funktion.

```
static int[] findPrimes(int n) {
   int[] res = new int[(n + 1) / 2];
   int lastIdx = 0;
   for (int i = 1; i <= n; i++)
      if (isPrime(i)) {
         res[lastIdx] = i;
         lastIdx++;
      }
   int[] realRes = new int[lastIdx];
   for (int i = 0; i < lastIdx; i++)
      realRes[i] = res[i];
   return realRes;
}
```

4. Aufgabe: Glücksspiel (Zusatzaufgabe)

Bei diesem Spiel gibt es *37* Felder, auf die der Spieler *200*€ (in Einheiten von 1€) verteilt. Danach wird eine Zufallszahl n ermittelt und an den Spieler wird der Gewinn ausgezahlt. Die Gewinnsumme errechnet sich wie folgt:

$$\sum_{i=0}^{36} (\texttt{board}[i] \cdot (6 - \mid i - n \mid))$$

wobei `board`$[i]$ den Wert der Belegung des Feldes i darstellt.
Allerdings sollen nur positive Summanden in die Summe eingehen.

4.1. Berechnung der Gewinnsumme Vervollständige die `play`-Funktion, welche bei der Eingabe eines *boards* die Gewinnsumme errechnet. Die Funktion soll also die Aufgabe eines Spielleiters bei einem Roulettespiel einnehmen.

Hinweis: Es wird sinnvoll sein, eine Methoden `static int abs(int x)` und eine Methode `static int max(int x, int y)` zu implementieren. `abs()` liefert dabei den Betrag der übergebenen Zahl zurück, während `max()` den größeren der beiden übergebenen Werte zurückliefert.

4.2. Ideale Boardbelegungen Erstelle ein paar Board-Belegungen und versuche Deinen Gewinn zu maximieren. Hierbei ist es nützlich, die Gewinnfunktion zu verstehen. Mache Dir also klar unter welchen Umständen man wieviel gewinnt.

- Gibt es Zahlen die günstiger sind als andere?
- Lohnt es sich auf einzelne Felder besonders viel zu setzen?
- Ist es besser den Einsatz gleichmäßig zu verteilen?

Versuche eine möglichst ideale Board-Belegung zu finden, indem du viele (z.B. 100 000) Spiele mit Deiner festen Belegung simulierst. Falls Du magst, kannst Du auch andere Gewinnfunktionen ausprobieren. z.B. $*36$ bei einem Volltreffer und $*6$ falls eines der acht benachbarten Felder getroffen wurde. Die Praktikabilität einer neuen Gewinnfunktion kannst Du mit der Lösung aus Aufgabe 3.1 überprüfen. Der durchschnittliche Gewinn sollte den Einsatz nicht überschreiten!

5. Aufgabe: Endlicher Automat (Zusatzaufgabe)

Definition Wort: Eine geordnete, nichtnegative Anzahl von Terminalsymbolen (auch Symbol oder Zeichen genannt). Definition Sprache: Menge von Worten (auch String oder Zeichenkette genannt). Eine Sprache kann unendlich groß sein. Trotzdem lassen sich einige Sprachen schematisch darstellen. $\underline{a}^\star$ bedeutet, dass das Terminalsymbol beliebig häufig (0,1,2. . . mal) vorkommt. Die Sprache dieser Aufgabe enthält also z.B. die Worte: (), (a), (bc), (d), (aabcd), (bcbcbcd), (aaddd). Nicht enthalten sind z.B. die Worte: λ (das leere Wort), (ab, (ab), (ada), (abd). Die Aufgabe ist nun, zu entscheiden, ob ein eingegebenes Wort (String) in dieser Sprache enthalten ist. Das Ergebnis ist also entweder: Ja, das eingebene Wort ist in der Sprache, oder: Nein, das eingegebene Wort ist nicht in der Sprache. Eine Grammatik ist eine Vorschrift, wie man Worte einer Sprache erzeugen kann. Außerdem sind alle Worte, die sich nicht durch die Grammatik ableiten lassen, nicht in der Sprache enthalten. Mit der gegebenen Grammatik lässt sich folgendermaßen ein Wort ableiten:

$$L = \left\{ \underline{(}\ \underline{a}^\star\ (\underline{b}\ \underline{c})^\star\ \underline{d}^\star\ \underline{)} \right\}$$

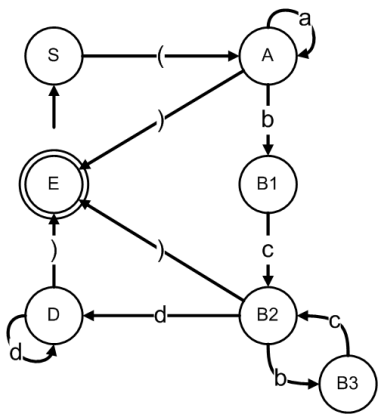

Abbildung 1: Ein Automat zu der Grammatik aus Aufgabe 4

Ableitung			Grammatik		
S	$\rightarrow$	$(A$	S	$\rightarrow$	$\underline{(}\ A$
	$\rightarrow$	$(aA$	A	$\rightarrow$	$\underline{a}\ A$
	$\rightarrow$	$(aB$		$\mid$	B
	$\rightarrow$	$(abcB$	B	$\rightarrow$	$\underline{b}\ \underline{c}\ B$
	$\rightarrow$	$(abcbcB$		$\mid$	D
	$\rightarrow$	$(abcbcD$	D	$\rightarrow$	$\underline{d}\ D$
	$\rightarrow$	$(abcbc)$		$\mid$	$\underline{)}$

5.1. Entscheider Gegegeben ist folgende Sprache L mit der o.g. Grammatik. Jedes Terminalsymbol ist ein `char`. Terminale sind unterstrichen, Nonterminale sind nicht unterstrichen.

Schreibe ein Programm, das für einen gegebenen String entscheidet, ob dieser String in der Sprache enthalten ist.

6. Aufgabe: Approximationen (Zusatzaufgabe)

6.1. Approximation von π Approximiere π mit der Formel von Ramanujan. Zerlege hierfür die gegebene Formel in die markierten Teilausdrücke und speichere die Zwischenergebnisse in temporären Variablen.

$$\pi \approx \underbrace{\frac{4}{\sqrt{522}}}\ \underbrace{\ln\left[\underbrace{\left(\frac{5+\sqrt{29}}{\sqrt{2}}\right)^3}\ \underbrace{\left(5\sqrt{29}+11\sqrt{6}\right)}\ \underbrace{\left(\underbrace{\sqrt{\frac{9+3\sqrt{6}}{4}}}+\underbrace{\sqrt{\frac{5+3\sqrt{6}}{4}}}\right)^6}\right]}$$

6.2. Approximiere e Entwickle eine iterative und eine rekusive Implementierung und vergleiche die Ergebnisse. Überlege Dir gegen welchen Wert .. konvergiert und setze diesen ein.

$$e - 2 \approx \frac{1+\frac{1+\frac{1+\frac{..}{6}}{5}}{4}}{3}}{2} = \frac{1}{2}(1+\frac{1}{3}(1+\frac{1}{4}(1+\frac{1}{5}(1+\frac{1}{6}(1+..)))))$$