PGdP Tutorium: Siebte Stunde

Benedikt Werner

München, 05. Dezember 2017

Hausaufgaben

- In kleinere Aufgaben einteilen
- Diese nacheinander implementieren
- Nach jedem Schritt testen!

Aufgabe 7.1 – Quicksort

1. *Tausch von Arrayelementen*: Schreiben Sie eine Java-Methode `void swap(int[] numbers, int i, int j)`, die in einem Array ganzer Zahlen `numbers` die Einträge mit den Indizes `i` und `j` vertauscht.
2. *Partitionierung eines Array-Ausschnitts*: Die Java-Funktion `int partition(int[] numbers, int left, int right)` soll zuerst den Wert `numbers[right]` als *Pivotwert* setzen. Danach sollen die Elemente des Arrays `numbers` zwischen den Positionen `left` und `right` *inklusive* der Pivotzelle derart vertauscht werden, dass links des linkesten Elementes mit dem *Pivotwert* nur kleinere Zahlen und rechts davon nur größere oder gleich große Zahlen stehen. Zurückgegeben wird der Index der bzw. einer Zelle, die danach den Pivotwert enthält.
3. Implementieren Sie mit Hilfe der bisher erarbeiteten Funktionen den Quicksort-Algorithmus rekursiv.
4. Erstellen Sie eine `main`-Routine, in der der Quicksort-Algorithmus für ein Array, dessen Elemente zufällig mit Zahlen belegt sind, getestet wird.
   Hinweis: Für $n > 0$ lässt sich eine Zufallszahl zwischen 0 und $n-1$ mit dem Befehl `int number = (int) (n*Math.random());` erzeugen.

Steck-Assembly

Instruktion	Immediate	Beschreibung
`NOP`	keins	Diese Instruktion hat keinerlei Effekt.
`ADD`	keins	Addiert o_1 und o_2
`SUB`	keins	Berechnet $o_1 - o_2$
`MUL`	keins	Multipliziert o_1 mit o_2
`MOD`	keins	Berechnet den Divisionsrest von o_1/o_2
`LDI`	v (16 Bit)	Lädt einen 16-Bit-Wert v auf den Stack; die oberen 16 Bit sollen den Wert 0 haben.
`LDS`	i (16 Bit)	Kopiert die durch i indizierte Frame-Zelle nach oben auf den Stack
`STS`	i (16 Bit)	Nimmt einen Wert vom Stack und speichert ihn in der durch i indizierten Frame-Zelle ab

Steck-Assembly

`JUMP`	i (16 Bit)	Springt zur Instruktion mit dem Index i
`JE`	i (16 Bit)	Springt zur Instruktion mit dem Index i, wenn $o_1 = o_2$ gilt; o_1 und o_2 werden vom Stack entfernt.
`JNE`	i (16 Bit)	Springt zur Instruktion mit dem Index i, wenn $o_1 \neq o_2$ gilt; o_1 und o_2 werden vom Stack entfernt.
`JLT`	i (16 Bit)	Springt zur Instruktion mit dem Index i, wenn $o_1 < o_2$ gilt; o_1 und o_2 werden vom Stack entfernt.

Steck-Assembly

`CALL`	n (16 Bit)	Ruft die Funktion an Index o_1 mit n Argumenten auf (siehe unten)
`RETURN`	n (16 Bit)	Kommt von einer Funktion zurück (siehe unten)
`IN`	keins	Liest einen Wert mittels `read()` ein und legt ihn auf den Stack
`OUT`	keins	Gibt o_1 mittels `write()` aus
`HALT`	keins	Beendet das Programm
`ALLOC`	n (16 Bit)	Reserviert Platz für n lokale Variablen; nur am Funktionsanfang gültig

ggt-Programm in Java

```java
public static int ggt(int a, int b) {
  if (b > a) {
    int temp = b;
    b = a;
    a = temp;
  }
  do {
    int temp = b;
    b = a % b;
    a = temp;
  } while(b != 0);
  return a;
}
```

```java
public static void main(String[] args) {
    int a = read();
    int b = read();
    write(ggt(a, b));
}
```

ggt-Programm in Steck-Assembly

```
IN
IN
LDI ggt
CALL 2
OUT
HALT

// a bei -1, b bei 0
ggt:
ALLOC 1
LDS -1
LDS 0
JLT loop
LDS 0
LDS -1
STS 0
STS -1
loop:
LDS 0
STS 1
LDS 0
LDS -1
MOD
STS 0
LDS 1
STS -1
LDS 0
LDI 0
JNE loop
LDS -1
// 2 Argumente und 1 lokale
// Variable freigeben
RETURN 3
```

Aufgabe 7.3 – Toolbox

Aufgabe 7.4 – Fakultät

Schreiben Sie ein Programm `Fak.java`, das die Fakultätsfunktion auf folgende Weisen berechnet.

1. In der Methode `public static int facRec(int n)` soll die Fakultätsfunktion *rekursiv* (aber nicht end-rekursiv) berechnet werden. Der Parameter `n` gibt dabei den Integer an, für den die Fakultät berechnet werden soll.
2. In der Methode `public static int facTailRec(int n)` soll die Fakultätsfunktion *endrekursiv* berechnet werden. Dafür benötigen Sie eine zusätzliche Hilfsmethode `private static int facTailRecHelper(int n, int k)`. Der Parameter `n` gibt dabei den Integer an, für den die Fakultät berechnet werden soll. Der Paramter `k` gibt das bisher berechnete Zwischenergebnis an.
3. In der Methode `facIt(int n, int k)` soll die Fakultätsfunktion *iterativ* durch Umwandlung der Endrekursion aus der Methode `facTailRec(int n, int k)` berechnet werden. D. h. die Berechnung erfolgt in einer `while(true){...}` Schleife.

Aufgabe 7.5 – Das Ende der Rekursion

- Gegeben sind die folgenden Funktionen $f : \mathbb{N} \to \mathbb{N}$ und $g : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$.

$$f(x) = g(x, 0)$$

$$g(x, y) = \begin{cases} x^y & x < 10 \\ (x\%10)^y + g(x/10, y + 1) & x \geq 10 \end{cases}$$

  Hierbei sei % der Modulo-Operator wie in Java und / der Operator für ganzzahlige Division ebenfalls wie in Java. D. h. $12345/10 = 1234$ und $12345\%10 = 5$.

  Erstellen Sie zunächst eine rekursive, aber nicht endrekursive Implementierung der Funktion f in der Methode `public static int frec(int x)`.

  Implementieren Sie die Funktion f dann *endrekursiv* in der Methode `public static int ftailrec(int x)`. Benötigen Sie dazu eine Hilfsfunktion?

  Verwenden Sie *keine* Schleifen und keine Membervariablen in Ihrer Implementierung.

- Entschleifen Sie die Partitionsfunktion aus der Quicksortaufgabe.