Institut für Informatik
Heinrich-Heine-Universität Düsseldorf
Prof. Dr. Alice McHardy
Robin Küppers
Sebastian Konietzny
Aaron Weimann

WS 2013/2014
26.11.2013
Übungsblatt 5

Einführung in die Informatik I
Übungen zur Vorlesung
Musterlösung

Die Lösungen der zu bearbeitenden Aufgaben müssen spätestens bis **Dienstag**, den **03.12.2013, 11:59 Uhr** über https://ilias.uni-duesseldorf.de abgegeben werden. Bitte achten Sie immer auf einen vollständigen und nachvollziehbaren Lösungsweg, da ansonsten Punkte abgezogen werden können! Vergessen Sie bitte nicht auf Ihrer Abgabe Ihren Namen und Ihre Matrikelnummer zu vermerken (bei Gruppenabgaben müssen sämtliche Mitglieder der Gruppe aufgeführt werden).

5.1 Ein- und Ausgabe (9 Punkte)

Aus der Vorlesung kennen Sie verschiedene Möglichkeiten die Ein- und Ausgabe eines `Java`-Programms umzulenken (z.B. Standard-Ausgabe eines Programms als Standard-Eingabe eines anderen Programms nutzen). In dieser Aufgabe sollen Sie dieses Wissen anwenden. Auf der Webseite finden Sie die Datei `b51input.txt`, die für diese Aufgabe als Eingabe benötigt wird. Zur Vereinfachung der Ein- und Ausgabe nutzen Sie bitte die Klassen `StdIn` und `StdOut`, die ebenfalls auf der Webseite zur Vorlesung (Material zu Vorlesung 4) zum Herunterladen angeboten werden. Die .java-Dateien für StdIn und StdOut müssen im selben Verzeichnis wie Ihre Programme liegen, damit der Compiler sie findet. Im Allgemeinen ist keine weitere Fehlerbehandlung vonnöten.

1. Schreiben Sie ein `Java`-Programm `Pipeline1`, das als Eingabe die Datei `b51input.txt` einliest. Das Programm soll solange von der Standard-Eingabe lesen, bis keine weiteren Daten zur Verfügung stehen. Nutzen Sie dafür die `StdIn`-Klasse. Die Datei `b51input.txt` enthält eine Reihe von hexadezimalen Werten (ein Wert pro Zeile), die von dem Programm `Pipeline1` ins Dezimalsystem überführt werden sollen. Schreiben Sie dazu eine Funktion `public static int hex2dec(String hex)`. Die umgewandelten Werte sollen auf die Standard-Ausgabe geschrieben werden. (4 Punkte)

   Tipp: Interpretieren Sie die hexadezimalen Werte als Zeichenkette. Die Methode `char charAt(int i)` gibt das Zeichen an Position `i` zurück. Für besonders eleganten Code dürfen Sie hier auch das Java-Konstrukt `switch` verwenden.

2. Schreiben Sie ein `Java`-Programm `Pipeline2`, das die Ausgabe von `Pipeline1` als Standard-Eingabe verwendet. Die gelesenen Dezimalwerte sollen als `ASCII`-Symbole interpretiert werden: Das Programm soll die Dezimalwerte in Zeichen konvertieren und auf die Standard-Ausgabe schreiben (2 Punkte).

   Tipp: Sie können einen `int`-Wert `i` nach `char` casten mit `(char)i`.

3. Geben Sie eine Pipeline an, die folgende Bedingungen erfüllt (2 Punkte):

   - Die Datei `b51input.txt` ist die Eingabe für `Pipeline1`.
   - Die Eingabe von `Pipeline2` ist die Ausgabe von `Pipeline1`.
   - Die Ausgabe von `Pipeline2` soll in eine Datei `b51output.txt` geschrieben werden.

4. Führen Sie die Pipeline aus. Was wird als Ergebnis in die Datei `b51output.txt` geschrieben? (1 Punkt)

Musterlösung

1.
```
public class Pipeline1 {

    private static int hex2dec(String hex) {
        int result = 0;
        for(int i = 0; i < hex.length(); i++) {
            int c = hex.charAt(i);
            int j = hex.length() - i - 1;
            if(c == '0')
                result += 0 * Math.pow(16,j);
            else if(c == '1')
                result += 1 * Math.pow(16,j);
            else if(c == '2')
                result += 2 * Math.pow(16,j);
            else if(c == '3')
                result += 3 * Math.pow(16,j);
            else if(c == '4')
                result += 4 * Math.pow(16,j);
            else if(c == '5')
                result += 5 * Math.pow(16,j);
            else if(c == '6')
                result += 6 * Math.pow(16,j);
            else if(c == '7')
                result += 7 * Math.pow(16,j);
            else if(c == '8')
                result += 8 * Math.pow(16,j);
            else if(c == '9')
                result += 9 * Math.pow(16,j);
            else if(c == 'A')
                result += 10 * Math.pow(16,j);
            else if(c == 'B')
                result += 11 * Math.pow(16,j);
            else if(c == 'C')
```

```
                result += 12 * Math.pow(16,j);
            else if(c == 'D')
                result += 13 * Math.pow(16,j);
            else if(c == 'E')
                result += 14 * Math.pow(16,j);
            else if(c == 'F')
                result += 15 * Math.pow(16,j);
        }
        return result;
    }



    public static void main(String[] args) {
        while (!StdIn.isEmpty()) {
            String hex = StdIn.readString();
            StdOut.println(hex2dec(hex));
        }
    }

}
```

2.
```
public class Pipeline2 {

    public static void main(String[] args) {
        while (!StdIn.isEmpty()) {
            int val = StdIn.readInt();
            StdOut.print((char) val);
        }
    }

}
```

3. `java Pipeline1 < b51input.txt | java Pipeline2 > b51output.txt`

4. `Output: Hello world.`

5.2 Rekursion (5 Punkte)

Von Übungsblatt 4 kennen Sie bereits die rekursive Definition der Fibonacci-Zahlen. Hier nochmal zur Erinnerung: Die Fibonacci-Folge $f_0, f_1, f_2, \ldots$ wird durch das folgende rekursive Bildungsgesetz definiert:

$$\begin{aligned} f_0 &= 0 \\ f_1 &= 1 \\ f_n &= f_{n-1} + f_{n-2} \text{ für } n \geq 2 \end{aligned}$$

1. Implementieren Sie die rekursive Variante der Fibonacci-Folge in *Java*. (2 Punkte)

2. Gegeben sind folgende rekursive Funktionen in Java:

```
public static int f(int x) {
    if (x == 0) return 1;
    else return f(x / 2) + g(x - 1);
}

public static int g(int x) {
    if (x == 0) return 2;
    else return g(x / 2) + f(x - 1);
}
```

   Bestimmen Sie den Funktionswert von `f(g(1))`. Geben Sie dabei die einzelnen Rekursionschritte an. (3 Punkte)

Musterlösung

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

2.

$f(g(1)) = \mathbf{?}$

Wir berechnen zunächst $g(1)$:

$$\begin{aligned} g(1) &= g(1) + f(0) \\ &= g(0) + f(0) \\ &= 2 + 1 \\ &= 3 \end{aligned}$$

Jetzt berechnen wir $f(g(1)) = f(3)$:

$$\begin{aligned} f(3) &= f(1) + g(2) \\ &= f(0) + g(0) + g(1) + f(1) \\ &= f(0) + g(0) + g(0) + f(0) + f(0) + g(0) \\ &= 1 + 2 + 2 + 1 + 1 + 2 \\ &= 9 \end{aligned}$$

$$\Rightarrow f(g(1)) = f(3) = 9$$

5.3 Rekursion II (9 Punkte)

1. Schreiben Sie ein `Java`-Programm mit einer Methode `mul`, die das Produkt aus einer natürlichen Zahl `i` und einer ganzen Zahl `g` mit einer rekursiven Hilfsmethode `mul_help` durch wiederholte Addition berechnet. Benutzen Sie dabei keine Schleifen. Geben Sie zunächst Rekursionsgleichungen für `mul` bzw. `mul_help` an. Prüfen Sie auch, dass das Programm keine ungültigen Eingaben erhält. (4 Punkte)

2. Der ganzzahlige Quotient `a div b` lässt sich aus `a` durch wiederholtes Subtrahieren von `b` bestimmen. Schreiben Sie ein Java-Programm mit einer Methode `div`, die den Quotienten mit einer rekursive Hilfsmethode `div_help` bestimmt. Geben Sie zunächst wieder die Rekursionsgleichungen für die Funktion `div` bzw. `div_help` an. Benutzen Sie dabei keine Schleifen. Prüfen Sie, dass das Programm keine ungültigen Eingaben erhält. (5 Punkte)

Musterlösung

1. Die Aufgaben können alternativ auch nach diesem Schema gelöst: $mul(g, i) = g + mul(g, i-1)$
   Rekursionsgleichungen:

$$\begin{aligned} \text{mul}(g, i) &= \text{mul\_help}(g, i, 0), \text{ falls } i \geq 0 \\ \text{mul\_help}(g, i, s) &= \text{mul\_help}(g, i-1, s+g) \\ \text{mul\_help}(g, 0, s) &= s \\ \text{mul\_help}(0, i, s) &= 0 \end{aligned}$$

```
public class Mul {

    public static void main(String[] args) {
        int g = Integer.parseInt(args[0]);
        int i = Integer.parseInt(args[1]);
        if(i < 0) System.out.println("i ist keine natürliche Zahl");
        else System.out.println(mul(g,i));
    }

    public static int mul (int g, int i) {
        return mul_help(g,i,0);
    }

    public static int mul_help (int g, int i, int s) {
        if(g == 0 || i == 0) return s;
        //Fall mul_help(0,i,s) = 0 ist hier mit abgedeckt
        else return mul_help(g, i-1, s+g);
    }

}
```

2. **Rekursionsgleichungen:** Die Aufgabenstellung

   Der ganzzahlige Quotient a div b impliziert, dass a `mod` $b = 0$ ansonsten müsste man die Rekursionsgleichungen anpassen. Im Code könnte man die Bedingung

   ```
   if (a==0) return s;
   ```

   mit

   ```
   if (a-b < 0) return s;
   ```

   ersetzen, um auch Division mit Rest zu realisieren.

$$\begin{aligned}
\text{div}(a, b) &= \text{div_help}(a, b, 0), \text{ falls } b \neq 0 \\
\text{div_help}(a, b, c) &= \text{div_help}(a - b, b, c + 1), \text{ falls } a \cdot b > 0 \\
\text{div_help}(a, b, c) &= \text{div_help}(a + b, b, c - 1), \text{ falls } a \cdot b < 0 \\
\text{div_help}(0, b, c) &= c
\end{aligned}$$

```
public class Div {

    public static void main(String[] args) {
        int a = Integer.parseInt(args[0]);
        int b = Integer.parseInt(args[1]);
        if(b == 0) System.out.println("Divisor muss ungleich 0 sein");
        else System.out.println(div(a,b));
    }

    public static int div(int a, int b) {
        int sign = Integer.signum(a*b);
        a = Math.abs(a);
        b = Math.abs(b);
        return div_help(a, b, 0)*sign;
    }

    public static int div_help (int a, int b, int s) {
        if (a==0) return s;
        else return div_help(a-b, b, s+1);
    }

}
```

5.4 Einfache Referenzen und Java (2 Punkte)

1. Welche Werte stehen nach der Ausführung des folgenden Codes in den Arrays `a` und `b`? Begründen Sie! (2 Punkte)

```
001    int[][] a;
002    int[][] b;
003
004    a = new int[2][2];
005    b = new int[2][2];
006
007    a[0][0] = 0;
008    a[0][1] = 1;
009    a[1][0] = 2;
010    a[1][1] = 3;
011    b[1] = a[0];
012    b[0] = b[1];
013    a[0][1] = 2;
```

Musterlösung

```
a = [[0, 2], [2, 3]]
b = [[0, 2], [0, 2]]
```

Begründung:

Das Array `a` (und natürlich auch `b`), sowie die Unterarrays von `a` und `b` werden zunächst mit 0en initialisiert. Anschließend wird Array `a` auf die Werte [[0, 1], [2, 3]] gesetzt. Danach wird `b[1]` auf `a[0]` referenziert. Das Array `b` enthält an dieser Stelle somit die Werte [[0, 0], [0, 1]]. Jetzt wird zusätzlich `b[0]` auf `b[1]` referenziert. Damit enthält `b` jetzt [[0, 1], [0, 1]], wobei beide Teilarrays (und `a[0]`) auf den selben Speicherbereich zeigen. Deswegen ändert sich der Inhalt von `b` auch in beiden Teilarrays, wenn `a[0][1]` auf 2 gesetzt wird. Aus dem gleichen Grund enthält `a` nun [[0, 2], [2, 3]].

5.5 Abgabe (x Punkte)

Geben Sie bitte unbedingt Sourcecode und Textantworten getrennt ab. Den Sourcecode als .java Dateien, eine Datei pro Programm, und die Textantworten in **einem** PDF.