Programmierstarthilfe SS 2008

Fakultät für Ingenieurwissenschaften und Informatik

3. Blatt

Für den 5. und 6.5.2008

Organisatorisches

Die Webseiten zur Veranstlatung sind unter `http://www.uni-ulm.de/in/mi/lehre/ss2008/programmierstarthilfe.html` zu finden.

Du kannst uns Fragen auch über die Adresse `programmierstarthilfe@lists.uni-ulm.de` stellen.

Wir stellen diesmal kurz unser Standardvorgehen für Aufgaben vor:

- Zuerst eine Idee entwickeln.
- Dann einen Algorithmus aufstellen, das ist meist etwas Pseudocode.
- Danach den Algorithmus in Programmcode übersetzen.
- Falls noch Fehler im Programm sind diese beseitigen.

1. Logik

Hin und wieder stößt man bei Aufgaben an Stellen, an denen ein einzelnes `if` nicht ausreicht. Zum Beispiel möchte ich in der Tageszeit-Aufgabe "guten Morgen" ausgeben, wenn es zwischen 8 und 11 Uhr ist. Dafür kann ich natürlich zwei `if`-Abfragen machen:

```
1 if (zeit >= 8)
2     if (zeit <= 11)
3         System.out.println("Guten Morgen!");
```

Oder aber ich verknüpfe die beiden Ausdrücke `zeit >= 8` und `zeit <= 11` zu einem gemeinsamen:

```
1 if (zeit >= 8 && zeit <= 11)
2     System.out.println("Guten Morgen!");
```

Dabei ist das `&&` ein logisches Und. Die Ausgabe findet also nur statt, wenn sowohl `zeit` ≥ 8 als auch `zeit` ≤ 11 gilt.

Folgende logische Verknüpfungen gibt es in Java:

- logisches Nicht: `!a` ergibt `false`, wenn `a` wahr ist und `true`, wenn `a` `false` ist

- logisches Und: `a && b` ergibt `true`, wenn sowohl `a` als auch `b` wahr sind
- logisches Oder: `a || b` ergibt `true`, wenn mindestens einer der beiden Ausdrücke `a` oder `b` wahr ist
- exclusives Oder: `a ^ b` ergibt `true`, wenn beide Ausdrücke einen unterschiedlichen Wahrheitswert haben

Diese Verknüpfungen kann ich jeweils auf einen (beim Nicht) bzw. zwei (bei allen anderen) boolsche Ausdrücke anwenden. Sinn macht zum Beispiel:

```
1 boolean a = false, b = true;
2 boolean wahrheit = !a;
3 boolean luege = a && b;
4 if (luege || wahrheit)
5     System.out.println("Das gilt immer.");
6 if (!((a ^ b) || luege && !b))
7     System.out.println("Gilt auch das immer?");
```

1.1. Erste Logikaufgaben

a) Schreibe ein Programm, das - ähnlich dem Beispiel oben - einige boolsche Variablen deklariert, ihnen Werte zuweist und diese dann Und-, Oder- und Nicht-verknüpft. Gib die Ergebnisse aus. Versuche ruhig mehrfache Verknüpfungen vorzunehmen.

b) Nimm nun einige Integer-Variablen hinzu und verknüpfe Vergleiche (z.B. <=, ==) dieser Variablen so, dass eine komplexere Bedingung abgefragt wird. Was tut der Vergleichsoperator `!=`?

c) Java nutzt verschiedene Prioritäten, wenn es einen Ausdruck auswertet. Bei `a+b*c` zum Beispiel, wird Punktrechnung vor Strichrechnung ausgeführt. Wie ist das bei boolschen Verknüpfungen? Finde heraus, ob `&&` oder ||stärker bindet. Das kannst du zum Beispiel tun, indem du den Ausdruck `a && b || c` nutzt. Wie ist es beim exklusiven Oder (`^`) und beim Nicht (`!`)?

1.2. Logische Verknüpfungen

Neben den einfachen logischen Operatoren und, oder und nicht gibt es auch weitere, die sich aus diesen zusammensetzen lassen.

- aus a folgt b = NICHT a ODER b
- a NOR b = NICHT (a ODER b)
- a XOR b = (a UND NICHT b) oder (NICHT a und b)
- a XNOR b = NICHT (a XOR b)

Definiere zwei boolesche Variablen, überprüfe die oben genannten Fälle und gib entsprechend Text aus wenn sie zutreffen (z.B. a XOR b trifft zu). Überprüfe deinen Code durch Ausprobieren verschiedener Belegungen.

1.3. Beweis über Wahrheitsbelegung

Für logische Ausdrücke `boolean a, b, c` gelten folgende Gesetze:

$$a \wedge b = b \wedge a$$

$$(a \wedge b) \vee c = (a \vee c) \wedge (b \vee c)$$

$$(a \oplus b) \wedge c = a \oplus b \oplus c$$

$$\neg(a \wedge b) = \neg a \vee \neg b$$

$$((\neg a \wedge (a \vee b)) \vee b) \wedge c = (a \wedge c) \vee (b \wedge c)$$

Dabei ist mit $\neg$ das Nicht, mit $\wedge$ das logische Und, mit $\vee$ das logische Oder und mit $\oplus$ das exclusive Oder gemeint.

Diese Gesetze kann man beweisen, indem man alle möglichen Kombinationen von wahren und falschen `a`, `b` und `c` einsetzt und jeweils überprüft, ob die Ergebnisse auf beiden Gleichungsseiten auch gleich sind. Schreibe ein Java Programm, dass alle Fälle ausprobiert und eine Warnung gibt, falls in einer Gleichung einer nicht stimmt (dann gilt die Gleichung nämlich nicht).

2. Mehr Schleifen

Eine Schleife die uns noch fehlt ist die do-while Schleife. Sie funktioniert wie die while Schleife, wird aber erst einmal durchlaufen und danach wird getestet ob sie weiter wiederholt werden soll. Hier soll so lange gewürfelt werden, bis der Würfel eine 6 ist.

```
1 int wuerfel;
2 do {
3   wuerfel = wuerfle(6);
4 } while (wuerfel!=6);
```

2.1. Schleifen ersetzen

Java verfügt über 3 Schleifenkonstrukte, von denen nur eines essentiell ist. Die folgende for-Schleife kannst du also durch eine while-Schleife oder durch eine do-while-Schleife ausdrücken, ohne dass das Programm ein anderes Ergebnis liefert. Demonstriere das.

```
1 for (int i = 0; i < 20; i++)
2     System.out.println("i = "+i+", i^2 = "(i*i));
```

2.2. Schleifen 3

a) Schreibe ein Programm, das mit Hilfe einer Schleife 9 Zeichen (`char`) ausgibt. Das erste und das letze Zeichen sollen ein `'o'` sein, der Rest Leerzeichen.

b) Ändere dein Programm so ab, dass die Zeile mit einer zweiten Schleife 9 mal ausgegeben wird. Dabei soll die Position des ersten `'o'` jede Zeile um eine Stelle weiter nach rechts wandern, das zweite `'o'` soll in entgegengesetzter Richtung laufen. Du solltest jetzt ein X aus `'o'`s sehen.

c) Ändere dein Programm so ab, dass das erste `'o'` durch ein `'\'` ersetzt wird, das zweite durch `'/'`. An der Stelle, an der sich die beiden überlappen soll ein `'X'` ausgegeben werden.

2.3. Schleifen 4

ROT13 (engl. rotate by 13 places, zu deutsch in etwa „rotiere um 13 Stellen“) ist eine Verschiebechiffre (auch Caesarchiffre genannt), mit der auf einfache Weise Texte verschlüsselt werden können. Dies geschieht durch Ersetzung von Buchstaben – bei ROT13 im speziellen wird jeder Buchstabe des lateinischen Alphabets durch den im Alphabet um 13 Stellen davor bzw. dahinter liegenden Buchstaben ersetzt. (aus *Wikipedia*)

Schreibe ein Programm, das folgenden Satz entschlüsselt:
`Vpu uno zvg qrz Obeqpbzchgre trfcebpura. - Haq? - Re unffg zvpu.`

Satzzeichen sind natürlich nicht verschlüsselt. Tipp: Benutze `c+=13` statt `c=c+13`. Bei `int` ist beides äquivalent, bei `char` funktioniert nur ersteres.

2.4. *Primzahltest

a) Schreibe ein Programm, welches testet ob es sich bei einer gegebenen Zahl um eine Primzahl handelt oder nicht. Die Zahl kann hardcodiert in deinem Programm stehen (d.h. sie muss nicht von der Eingabe eingelesen werden). Überlege dir dazu zunächst, wann eine Zahl prim ist und wie du das nachprüfen kannst.

b) Schreibe dein Programm so um, dass es alle Primzahlen zwischen 0 und 100 ausgibt.