

**Aufgabe 1 Zweierkomplementdarstellung**

(6 = 2 + 4 Punkte)

1. Stellen Sie folgende vierstellige Zweierkomplementzahlen als Dezimalzahlen dar:

 $(1101)_2$  und  $(0101)_2$ .**Lösungsvorschlag:** $(1101)_2 = -3$ ,  $(0101)_2 = 5$ .

2. Ganze Zahlen in Zweierkomplementdarstellung kann man in Java als ein Feld von Booleschen Werten repräsentieren. Dabei wird folgende Konvention getroffen:  $n$ -stellige Zweierkomplementzahlen  $(b_1 \dots b_n)_2$  werden durch Felder der Länge  $n$  repräsentiert, sodaß im  $i$ -ten Feldeintrag die  $(i + 1)$ -te Ziffer von links  $b_{i+1}$  steht ( $0 \leq i \leq n - 1$ ). So wird etwa  $-4 = (1100)_2$  repräsentiert durch

```
boolean[] minusfour = { true, true, false, false };
```

Implementieren Sie eine Java-Funktion

```
public static int twocompval(boolean[] c)
```

die den Wert einer Zweierkomplementzahl  $c$  berechnet.**Lösungsvorschlag:**

```
public static int twocompval(boolean[] c) {
    int result = 0;
    int k = 1;

    for (int i = c.length-1; i >= 0; i = i-1) {
        if (c[i])
            result = result+k;
        k = k*2;
    }
    if (c[0])
        result = result-k;
    return result;
}
```

**Aufgabe 2 BNF-Grammatik**

(6 = 4 + 2 Punkte)

Beispiele für die Deklaration zweidimensionaler `boolean`-Feldes mit Initialisierung in Java — so wie aus der Vorlesung bekannt — sind etwa:

```
boolean[] [] x={{true,true,true},{true,false}};
boolean[] [] y={{true,false},{true,true,false}};
boolean[] [] xy={{true},{},{false}};
```

Die Feldinitialisierung kann in beiden Felddimensionen beliebig viele Elemente enthalten. Wir nehmen an, daß der Name des Feldes nur aus Kleinbuchstaben („a“-„z“) besteht.

1. Geben Sie eine BNF-Grammatik für solche Java-Deklarationen von zweidimensionalen `boolean`-Feldes mit Initialisierung an, wobei `boolean[] []`, `=`, `{`, `}`, `;`, `,`, `"a"`, `...`, `"z"`, `true` und `false` die Terminalsymbole der Grammatik sind. Zwischen den Terminalsymbolen sind keine Leerzeichen vorzusehen.

**Lösungsvorschlag:**

Mit dem Startsymbol *ArrayDecl* beschreibt

```
ArrayDecl = "boolean[] [] " Name "=" Init ";"
Name = "a" | ... | "z" {"a" | ... | "z"}
Init = "{" "}" | "{" Init2 {"," Init2} "}"
Init2 = "{" "}" | "{" Literal {""," Literal}* "}"
Literal = "true" | "false"
```

das Gewünschte.

2. Geben Sie eine Ableitung an für: `boolean[] [] xy={{true},{}}`;

**Lösungsvorschlag:**

Eine mögliche Ableitung ist:

```
ArrayDecl
→ "boolean[] [] " Name "=" Init ";"
→ "boolean[] [] " "xy" "=" Init ";"
→ "boolean[] [] " "xy" "=" "{" Init2 {""," Init2}* "}" ";"
→ "boolean[] [] " "xy" "=" "{" Init2 "," Init2 "}" ";"
→ "boolean[] [] " "xy" "=" "{" "{" Literal {""," Literal}* "}" "," Init2 "}" ";"
→ "boolean[] [] " "xy" "=" "{" "{" Literal "}" "," Init2 "}" ";"
→ "boolean[] [] " "xy" "=" "{" "{" "true" "}" "," "{" "}" "}" ";"
```

**Aufgabe 3 Objektorientierung**

(10 = 3 + 4 + 3 Punkte)

Eine Firma hat mehrere Warenlager, aber mindestens eines. Die Warenlager halten jeweils mehrere Produkte vorrätig, aber mindestens eines. Produkte haben einen Namen, eine Nummer und einen Preis. Jacken und Hosen sind Produkte. Jacken und Hosen haben eine Größe.

Erstellen Sie eine Java-Implementierung für diese Situation, die den folgenden Anforderungen genügt:

- Die Implementierung stellt eine geeignete Klassenhierarchie zur Verfügung. Firmen haben ein Attribut mit der Anzahl ihrer Warenlager. Warenlager haben ein Attribut mit der Anzahl der in ihnen geführten Produkte. Produktnamen werden durch den Datentyp **String** repräsentiert, Produktnummern durch **int**, Preise durch **int** und Größen durch **char**.  
Es müssen keine Konstruktoren angegeben werden.
- Es gibt eine Instanzmethode, die es ermöglicht, zu einer Produktnummer in einem Lager den Preis zu ermitteln. Die Methode soll  $-1$  zurückliefern, wenn kein Produkt mit dieser Nummer im Lager vorhanden ist. Es darf davon ausgegangen werden, daß ein Lager keine zwei Produkte mit gleicher Nummer vorrätig hält.
- Es gibt eine Instanzmethode, die es ermöglicht, zu einer Produktnummer dasjenige Lager einer Firma zu finden, das ein Produkt mit dieser Nummer zum minimalen Preis vorrätig hat. Dabei darf davon ausgegangen werden, daß mindestens ein Lager der Firma ein Produkt mit dieser Nummer vorrätig hat.
- Die geforderten Methoden müssen in den passenden Klassen deklariert sein.

**Lösungsvorschlag:**

```
class Company {
    Store[] stores;
    int numstores;

    Store findmin(int num) {
        Store result = stores[0];
        int min = -1;

        for (int i = 0; i < numstores; i++) {
            int price = stores[i].find(num);
            if (price >= 0) {
                if (min == -1)
                    min = price;
                else
                    if (price < min) {
                        result = stores[i];
                        min = price;
                    }
            }
        }
        return result;
    }
}
```

```
class Store {
    Product[] products;
    int numproducts;

    int find(int num) {
        for (int i = 0; i < numproducts; i++) {
            if (products[i].num == num)
                return products[i].price;
        }
        return -1;
    }
}

class Product {
    String name;
    int num;
    int price;
}

class Jacket extends Product {
    char size;
}

class Trousers extends Product {
    char size;
}
```

**Aufgabe 4 Listen**

(8 = 3 + 5 Punkte)

Listen aus ganzen Zahlen können in Java — wie aus der Übung bekannt — als Objektstrukturen aus Instanzen der folgenden Klasse repräsentiert werden:

```
class List {
    boolean isempty;
    int contents;
    List next;

    public List() {
        isempty = true;
    }

    public List(int c, List n) {
        contents = c;
        next = n;
        isempty = false;
    }
}
```

Die folgende iterative Methode der Klasse `List` spaltet die aktuelle Liste in zwei Teillisten auf: Die Methode liefert ein Feld der Länge zwei zurück. Im ersten Eintrag des zurückgegebenen Feldes steht die Teilliste aus genau all denjenigen Elementen der ursprünglichen Liste, die kleiner oder gleich dem Wert von `x` sind; im zweiten Eintrag des zurückgegebenen Feldes aber die Teilliste aus allen restlichen Elemente der ursprünglichen Liste. Dabei wird die Reihenfolge der Elemente erhalten werden. Es wird die Hilfsmethode `List append(List l)` aus den Übungen in der Klasse `List` vorausgesetzt, die die Liste `l` an die aktuelle Liste anhängt.

Die Methode `List[] split(int x)` liefert also zum Beispiel für die Liste `[3, 9, 7, 4, 12, 1, 27]` und für `x = 7` das Paar der Listen `[3, 7, 4, 1]` und `[9, 12, 27]` als Ergebnis.

```
List[] split(int x) {
    List[] ret = new List[2];
    ret[0] = new List();
    ret[1] = new List();

    int c = contents; List n = next; boolean e = isempty;
    for (; e == false; c = n.contents, e = n.isempty, n = n.next) {
        if (c <= x)
            ret[0] = ret[0].append(new List(c, new List()));
        else
            ret[1] = ret[1].append(new List(c, new List()));
    }
    return ret;
}
```

1. Geben Sie die Worst-case-Zeitkomplexität dieser Methode `List[] split(int x)` an, wobei die Länge der Liste, für die die Methode aufgerufen wird, als Komplexitätsparameter betrachtet wird. Die Methode `List append(List l)` hat die Worst-case-Zeitkomplexität  $O(n)$ , wobei  $n$  die Länge der Liste bezeichnet, für die `append` aufgerufen wird. Begründen Sie Ihre Antwort.

**Lösungsvorschlag:**

Im schlechtesten Fall ist  $x$  größer oder kleiner als alle Elemente der Liste, für die `split` aufgerufen wird. Ist die Länge dieser Liste  $n$ , so wird die `for`-Schleife  $n$  mal durch-

laufen, in den `append`-Aufrufen werden jeweils  $O(n)$  Schritte benötigt und damit ist die Gesamtkomplexität im schlechtesten Fall  $O(n^2)$ . Genauer benötigen die `append`-Aufrufe  $O(1 + 2 + \dots + n) = O(n \cdot (n + 1)/2) = O(n^2)$  Schritte.

2. Implementieren Sie eine rekursive Methode `List[] splitrec(int x)` für die Klasse `List`, die das gleiche leistet wie `List[] split(int x)`. (*Hinweis*: Die Methode `append` muß nicht verwendet werden.)

**Lösungsvorschlag:**

```
public List[] splitrec(int x) {
    if (isempty) {
        List[] s = new List[2];
        s[0] = new List();
        s[1] = new List();
        return s;
    }
    List[] s = next.splitrec(x);
    if (contents <= x)
        s[0] = new List(contents, s[0]);
    else
        s[1] = new List(contents, s[1]);
    return s;
}
```