

## Übungsblatt 8

# IMPERATIVE PROGRAMMIERUNG

**Abgabe: bis 17.12.2004, 13:00 Uhr in die Einwurfkästen im Untergeschoß des neuen Informatikgebäudes am Fasanengarten**

**Erreichbare Punkte: 35 P / 25 T**  
**(Praktische Aufgaben / Theoretische Aufgaben)**

### **Hinweis:**

Alle Programmieraufgaben sind unter Verwendung der in der Vorlesung vorgestellten In-/Out-Klassen zu erstellen und (ähnlich wie die Beispielprogramme im Skript) in ausführlicher Form zu kommentieren.

## **1 ZUSAMMENGESetzte ANWEISUNGEN**

### **1.1 Schleifenvergleich (12P/3T)**

Schreiben Sie für die folgenden Aufgaben jeweils ein Programm, das den am besten geeigneten Schleifentyp (while-, for- oder do-while-Schleife) verwendet. Begründen Sie Ihre Auswahl.

- a) Zuerst wird ein positiver ganzzahliger Wert eingelesen. Es werden anschließend genau so viele Zahlen eingelesen, wie der anfangs eingelesene Wert angibt und nach Eingabe jeder Zahl wird die Summe der bis dahin gelesenen Zahlen ausgegeben. (4P/1T)
- b) Bis zur Eingabe einer 0 sind ganze Zahlen einzulesen. Sofern es sich bei der Eingabe um keine 0 handelt, wird nach Eingabe die Summe der bisher gelesenen Zahlen ausgegeben. (4P/1T)
- c) Zunächst ist ein positiver ganzzahliger Grenzwert einzulesen. Danach werden ganze Zahlen eingelesen und nach Eingabe jeder Zahl wird die Summe der bisher gelesenen Zahlen ausgegeben, bis die Summe (inklusive der zuletzt eingegebenen Zahl) den Grenzwert überschritten hat. (4P/1T)

### **1.2 Schleifentransformation (4T)**

Wandeln Sie folgende for-Schleife in eine while-Schleife und in eine do-while-Schleife um. Verwenden Sie in den Schleifen keine break-Anweisung.

```
int s = 0;
for (;;) {
    int x = In.readInt();
    if (x<0) break;
    s = s + x;
}
```

### 1.3 Schleifeninvariante (15T)

Gegeben sei folgendes Programmstück:

```
int a = In.readInt();           // read two integer values
int b = In.readInt();
if (a > b) {                   // if a > b exchange a and b
    int h = a;
    a = b;
    b = h;
}
// Assertion a <= b
int z = b;                     // start the loop with the higher integer value as z
int i = 1;
// Assertion (z == b) && (i == 1)
while ((z % a) != 0){         // while a is no divisor of z loop
// Assertion when entering loop (z % a) != 0
// position 1
    z += b;
// position 2
    i++;
// position 3
}
// Assertion after leaving the loop z = LCM(a,b)
// LCM is the abbreviation to least common multiple
```

Beweisen oder widerlegen Sie die Korrektheit der folgenden Aussagen:

- $z == b * i$  ist eine Schleifeninvariante für die while-Schleife. (7T)
- $LCM(a,b) == LCM(a,z)$  ist eine Schleifeninvariante für die while-Schleife. (5T)
- Die while-Schleife terminiert. (3T)

#### Hinweis:

Nutzen Sie geeignete Zusicherungen an den Stellen position 1, position 2 und position 3, um die Beweise zu führen.

## 2 METHODEN

### 2.1 Wiederverwendung von Programmcode (4P)

#### Würfelsimulator III

Ihr Programm aus Aufgabe 2.2 des 7. Übungsblatts soll um eine Ausgabe der Häufigkeiten erweitert werden.

Erweitern Sie Ihr Programm hierzu um eine Methode, die die relative Häufigkeit eines Intervalls grafisch darstellt. Geben Sie unter Verwendung dieser Methode die berechneten Häufigkeiten in Ihrem Programm aus.

Für die Formatierung der Ausgabe soll eine maximale Breite von 60 Zeichen für die Normierung verwendet werden.

Beispieldarstellung:

Häufigkeiten:

=====

```

1 | *****
2 | *****
3 | *****
4 | *****
5 | *****
6 | *****

```

## 3 DATENSTRUKTUREN

### 3.1 Lotto (4P)

Schreiben Sie ein Programm mit dem Namen `Lotto`, dessen Methode `void drawing (int[] numbers)` die 6 Zahlen aus 49 zieht, in einem Array speichert und ungeordnet ausgibt.

**Hinweis:**

Benutzen Sie zum Erzeugen der Zufallszahlen die Methode `Math.random()`.

Beachten Sie: Es sollen 6 **verschiedene** Zahlen gezogen werden.

## 4 DATENSTRUKTUREN

### 4.1 Schiffe versenken (15P/3T)

In dieser Aufgabe soll das Spiel Schiffe versenken programmiert werden. Das Spiel funktioniert so: Jeder Spieler hat vor sich ein Blatt Papier mit zwei gleich großen Spielfeldern. In einem trägt er vor dem Beginn des Spieles – unsichtbar für den Gegner – die Positionen seiner eigenen Schiffe ein. Auf dem anderen werden die während des Spieles ermittelten Ergebnisse von Schüssen auf die feindliche Flotte vermerkt. Abwechselnd gibt ein Spieler dem anderen ein Koordinatenpaar an, und dieser antwortet mit **Treffer** oder **Wasser**, je nachdem, ob er an der angegebenen Stelle ein Schiff platziert hatte oder nicht.

Schreiben Sie ein Programm, das Schiffe versenken mit Ihnen spielt. Zunächst soll das Spiel nur in eine Richtung gehen: Der Computer verwaltet eine Flotte und gibt auf Ihre Fragen nach Schiffen Antworten. Das 8 x 8 Felder große Spielfeld soll wie folgt belegt werden:

|   | a | b | c | d | e | f | g | h |
|---|---|---|---|---|---|---|---|---|
| 1 |   |   |   |   |   | X |   |   |
| 2 |   | X | X | X | X |   |   |   |
| 3 |   |   | X |   |   | X |   |   |
| 4 | X |   |   |   |   | X |   |   |
| 5 |   |   |   |   |   | X |   |   |
| 6 |   | X | X | X |   |   |   |   |
| 7 |   | X | X |   |   |   |   |   |
| 8 |   |   |   |   |   |   |   |   |

Überlegen Sie sich, welche Datenstruktur am besten geeignet ist, um das Spielfeld zu modellieren. Überlegen Sie sich weiterhin eine geeignete Kodierung für den Zustand eines Feldes (Schiff oder Wasser).

Nun soll der Rechner solange

- ein Koordinatenpaar einlesen und
  - das Ergebnis (Schiff oder Wasser) ausgeben,
- bis alle Schiffe versenkt wurden. Beispiel für einen Schuss:

Spalte (a-h)? c

Zeile (1-8)? 6

Treffer !!!

Alle Schiffe versenkt.

>

- a) Überlegen Sie sich zuerst, was Sie programmieren müssen, bevor Sie loslegen. Schreiben Sie sich die einzelnen Arbeitsschritte, die Ihr Programm ausführen muss, als UML-Aktivitätsdiagramm auf. (3T)
- b) Implementieren Sie die Programmschritte, die Sie in Teilaufgabe a) angegeben haben, und dokumentieren Sie die einzelnen Schritte ausreichend. (15P)