Grundlagen der Programmkonstruktion
Übungsblatt 5

Bitte beachten Sie, dass es nicht ausreicht, wenn sie eine Lösung an der Tafel zwar aufschreiben, aber nicht erklären können.

1 Zusicherungen, Termination und statisches Programmverständnis (1.5)

Gegeben ist die folgende Methode `year(int days)`

```
public static boolean isLeapYear(int year) {
    // Gibt true zurueck, wenn year ein Schaltjahr ist,
    // ansonsten false.
}

public static int year (int days) {
    // Gibt die aktuelle Jahreszahl zurueck, wenn
    // seit dem 1.1.1980 "days" Tage vergangen sind.

    assert days >= 0;

    int year = 1980;

    while (days > 365) {
        if (IsLeapYear(year)) {
            if (days > 366) {
                days -= 366;
                year += 1;
            }
        } else {
            days -= 365;
            year += 1;
        }
    }

    return year;
}
```

1.1 Zusicherung (0.5)

Finden Sie Schleifeninvarianten, welche die möglichen Wertebereiche aller lokalen Variablen eingrenzen.

1.2 Termination I (0.5)

Die Methode `years(int days)` ist fehlerhaft: Für bestimmte Eingaben terminiert die `while`-Schleife nicht. Finden Sie den Fehlerfall.

1.3 Termination II (0.5)

Korrigieren Sie die `while`-Schleife, sodass sie immer terminiert und ein korrektes Ergebnis zurückliefert.

2 Iterator (1.5)

Erstellen Sie für die Klasse `IntHashtable` (Skriptum, Seite 269, Listing 4.18) einen Iterator. Dieser soll (in beliebiger Reihenfolge) alle Elemente in der Hashtable zurückliefern. Die Methode `remove()` soll dabei undefiniert bleiben.

Hinweise

Sie haben bereits in der Vorlesung einen Iterator für eine Liste kennengelernt. Gehen Sie von diesem aus um zu ihrer Lösung zu gelangen. Sie können sich in `IntHashtable` eine Methode (mit default-Sichtbarkeit) `IntListNode[] getTab()` definieren, um auf das Array in `IntHashtable` zugreifen zu können.

3 GenQueue (2)

3.1 Erstellen der Datenstruktur (2)

Eine Queue ist eine Datenstruktur, die zwei Methoden anbietet, vergleichbar mit einem Stack. Diese Methoden heißen `enqueue` und `dequeue`. `enqueue` fügt ein Element der Queue hinzu (wie die Methode `push` bei einem Stack). `dequeue` löscht ein Element von der Queue und gibt dieses Element zurück (ähnlich wie die Methode `pop` bei einem Stack). Im Gegensatz zu einem Stack gibt die Methode `dequeue` NICHT das zuletzt auf den Stack gelegte Element zurück, sondern das Element, das schon am längsten in der Queue enthalten ist. Manchmal bezeichnet man eine Queue auch als FIFO (first in, first out) und einen Stack als LIFO (last in, first out).

Erstellen Sie eine generische Klasse `GenQueue` mit einem Typparameter, der den Typ der Elemente in der Queue bestimmt. Die Klasse soll eine Queue (für eine unbeschränkte Anzahl an Elementen) repräsentieren und einen Konstruktor beinhalten, der eine leere Queue erzeugt. Den Fehlerfall, dass `dequeue` auf eine leere Queue ausgeführt wird, müssen Sie nicht behandeln.

Sie dürfen für die Implementierung von `GenQueue` keine anderen schon fertig implementierten Datenstrukturen verwenden, insbesondere keine Arrays und Collection-Klassen aus den Java-Bibliotheken!

Beispiel

```
GenQueue<String> queue = new GenQueue<String>();

queue.enqueue("A");
queue.enqueue("B");

System.out.println(queue.dequeue()); // "A"
System.out.println(queue.dequeue()); // "B"
```