Prof. Dr. Uwe Schmidt

fhwedel
UNIVERSITY OF APPLIED SCIENCES

2. Februar 2010

---

Aufgaben zur Klausur **C** und **Objektorientierte Programmierung** im WS 2009/10 (WI h103, II h105, MI h353)

Zeit: 150 Minuten erlaubte Hilfsmittel: keine

Bitte tragen Sie Ihre Antworten und fertigen Lösungen ausschließlich an den gekennzeichneten Stellen in das Aufgabenblatt ein. Ist ihre Lösung wesentlich umfangreicher, so überprüfen Sie bitte nochmals Ihren Lösungsweg.

Sollten Unklarheiten oder Mehrdeutigkeiten bei der Aufgabenstellung auftreten, so notieren Sie bitte, wie Sie die Aufgabe interpretiert haben.

Viel Erfolg!

Diese Klausur besteht einschließlich dieses Deckblattes aus 16 Seiten.

---

**Aufgabe 1:**

Gegeben seien die folgenden sieben Hash-Funktionen für Strings:

```
typedef unsigned int Hash;
typedef char *String;

Hash hash1(String s) {
    return *s;
}

Hash hash2(String s) {
    Hash res;
    for (res = 0; *s; res += *s, ++s);
    return res;
}

Hash hash3(String s) {
    return (Hash) s;
}

Hash hash4(String s) {
    Hash res;
    for (res = 0; *s; res = 31 * res + *s, ++s);
    return res;
}

Hash hash5(String s) {
    Hash res;
    for (res = 0; *s; res >>= 5, res += *s, ++s);
    return res;
}

Hash hash6(String s) {
    Hash res;
    for (res = 1; *s; res *= *s, ++s);
    return res;
}

Hash hash7(String s) {
    Hash res;
    for (res = 0; *s; res = (res << 5) + *s - res, ++s);
    return res;
}
```

1. Welche dieser Funktionen erfüllen nicht die funktionalen Anforderungen an eine Hash-Funktion?

   hash1 ☐ hash2 ☐ hash3 ☐ hash4 ☐ hash5 ☐ hash6 ☐ hash7 ☐

2. Welche dieser Funktionen arbeiten bei der Berechnung mit undefinierten Werten, liefern also zufällige Resultate?

   hash1 ☐ hash2 ☐ hash3 ☐ hash4 ☐ hash5 ☐ hash6 ☐ hash7 ☐

3. Welche dieser Funktionen sind als Hash-Funktionen ungeeignet, da die Hash-Werte sehr ungleichmäßig häufig getroffen werden?

   hash1 ☐ hash2 ☐ hash3 ☐ hash4 ☐ hash5 ☐ hash6 ☐ hash7 ☐

4. Welche dieser Funktionen sind als Hash-Funktionen ungeeignet, da sie *ähnliche* Werte auf gleiche Hash-Werte abbilden?

   hash1 ☐ hash2 ☐ hash3 ☐ hash4 ☐ hash5 ☐ hash6 ☐ hash7 ☐

5. Welche Funktionen sind aus Effizienzgründen als Hash-Funktion ungeeignet?

   hash1 ☐ hash2 ☐ hash3 ☐ hash4 ☐ hash5 ☐ hash6 ☐ hash7 ☐

6. Welche Funktionen sind für die effizente Implementierung von Hash-Tabellen geeignet?

   hash1 ☐ hash2 ☐ hash3 ☐ hash4 ☐ hash5 ☐ hash6 ☐ hash7 ☐

---

**Aufgabe 2:**

Gegeben sei das folgende C-Programmstück:

```
#include <stdio.h>

typedef char * Element;

typedef struct Node * Tree;
struct Node {
  Element info;
  unsigned int arity;
  Tree * children;
};

typedef void (*ProcessElement)(Element e);

void printElement(Element e) {
  printf("%s\n", e);
}

void printTree(Tree t) {
  printElement(t-> info);
  {
    unsigned int i;
    for (i = 0; i < t->arity; ++i)
      printTree(t->children[i]);
  }
}
```

In diesem Programmteil wird eine Datenstruktur für n-stellige Bäume definiert. Die an den Knoten gespeicherte Information ist in diesem Beispiel ein Text. Die Funktion *printTree* traversiert einen Baum und gibt alle Texte an den besuchten Knoten aus.

Entwickeln Sie eine Funktion *void processTree(Tree t, ... )*, die das Traversieren einen Baumes auf die gleiche Art macht, wie *printTree*, die aber mit der an den Knoten auszuführenden Aktion parametrisiert ist:

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Reimplementieren Sie *printTree* mit *processTree*:

. . . . . .

. . . . . .

. . . . . .

Entwicken Sie eine Funktion *unsigned int card(Tree t)*, die die Anzahl der Knoten in einem Baum zählt. Implementieren Sie diese Funktion mit Hilfe von *processTree*:

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

---

**Aufgabe 3:**

In dieser Aufgabe geht es darum, die Laufzeit von Operationen für verschiedene Datenstrukturen abzuschätzen. Die Laufzeit von Operationen auf Listen und Bäumen hängt üblicherweise von der Anzahl der Elemente in einer Liste oder in einem Baum ab. In dieser Aufgabe sollen $n, n_1, n_2, ...$ die Anzahl der Elemente in den Listen $l, l_1, l_2, ...$ oder Bäumen $b, b_1, b_2, ...$ bezeichnen. Verwenden Sie bitte die *Groß–O–*Notation.

1. Laufzeitkomplexität für das Einfügen eines Elementes $e$ in eine sortierte, einfach verkettete Liste ohne Duplikate: $insert(e, l)$

   .....

2. Laufzeitkomplexität für das Einfügen eines Elementes $e$ in eine sortierte, einfach verkettete Liste mit Duplikaten: $insert(e, l)$

   .....

3. Laufzeitkomplexität für das Einfügen eines Elementes $e$ in eine unsortierte, einfach verkettete Liste mit Duplikaten: $insert(e, l)$

   .....

4. Laufzeitkomplexität für das Konkatenieren zweier einfach verketteter Listen: $concat(l_1, l_2)$

   .....

5. Laufzeitkomplexität für das Konkatenieren zweier einfach verketteter, als Ring implementierter Listen: $concat(l_1, l_2)$

   .....

6. Laufzeitkomplexität für das Anhängen eines Elementes $e$ am Ende einer einfach verketteten Liste: $append(l, e)$

   .....

7. Laufzeitkomplexität für das Mischen aller Elemente zweier sortierter, einfach verketteter Listen ohne Duplikate: $merge(l_1, l_2)$

   .....

8. Laufzeitkomplexität für das Mischen aller Elemente zweier unsortierter, einfach verketteter Listen ohne Duplikate: $merge(l_1, l_2)$

   .....

9. Laufzeitkomplexität für das Suchen eines Elementes $e$ in einer sortierten, einfach verketteten Liste mit Duplikaten: $isIn(e, l)$

. . . . . .

10. Laufzeitkomplexität im Mittel für das Suchen eines Elementes $e$ in einem Rot-Schwarz-Baum: $isIn(e, b)$

. . . . . .

11. Laufzeitkomplexität im Mittel für das Suchen eines Elementes $e$ in einer binären Halde: $isIn(e, b)$

. . . . . .

12. Laufzeitkomplexität im schlechtesten Fall für das Suchen eines Elementes $e$ in einem binären Suchbaum: $isIn(e, b)$

. . . . . .

13. Laufzeitkomplexität im Mittel für das Einfügen eines Elementes $e$ in einen binären Suchbaum: $insert(e, b)$

. . . . . .

14. Laufzeitkomplexität im schlechtesten Fall für das Einfügen eines Elementes $e$ in einen Rot-Schwar-Baum: $insert(e, b)$

. . . . . .

15. Laufzeitkomplexität im Mittel für das Einfügen eines Elementes $e$ in eine binäre Halde: $insert(e, b)$

. . . . . .

16. Laufzeitkomplexität für das Anhängen eines Elementes $e$ am Ende einer einfach verketteten, als Ring implementierter Liste: $append(l, e)$

. . . . . .

17. Laufzeitkomplexität für das Einfügen eines Elementes $e$ in eine sortierte, einfach verkettete, als Ring implementierte Liste: $insert(e, l)$

. . . . . .

**Aufgabe 4:**

Bitte lesen Sie diese Aufgabe vollständig durch, bevor Sie beginnen, die einzelnen Lösungsteile zu entwickeln.

In dieser Aufgabe soll ein einfacher Baukasten für die Verarbeitung von Zahlenfolgen entwickelt werden. Zahlenfolgen besitzen folgende Schnittstellen:

```
interface Sequence extends Duplicate {
    double next();
}

interface Duplicate {
    Duplicate dup();
}
```

Die Zahlenfolgen werden wie Datenströme verarbeitet, es gibt eine Methode zur Berechnung des nächsten Elements in der Folge. Die hier verwendete Technik findet sich zum Beispiel in der Klasse Reader im Paket java.io und deren Unterklassen wieder, insbesondere in den Filterklassen. Da hier mit unbeschränkt langen Zahlenfolgen gearbeitet wird, gibt es im Gegensatz zu Eingabe–Datenströmen keinen Test auf Ende der Zahlenfolge. Zusätzlich soll es über die Schnittstelle Duplicate möglich sein, eine neue Zahlenfolge zu erzeugen, die die gleiche Folge liefert wie die Ausfolge, unabhängig davon, wieviele Elemente schon aus der Ausgangsfolge ausgelesen wurden.

Um Zahlenfolgen zu erzeugen, gibt es einige einfache Klassen. Mit der Klasse *Const* können konstante Zahlenfolgen generiert werden.

```
public
class Const implements Sequence {
    private final double value;

    public Const() { this(0); }
    public Const(double value) { this.value = value; }

    public double next() { return value; }

    public Duplicate dup() {

        . . . . . .

        . . . . . .
    }
}
```

Vervollständigen Sie die Klasse.

Mit *Count* kann gezählt werden ($0, 1, 2, ...$ oder $1, 2, 3, ...$)

```
public class Count implements Sequence {
    private double cnt;

        . . . . . .

    public Count() { this(0); }
    public Count(double start) {

        . . . . . .

        . . . . . .
    }
    public double next() {
        return cnt++;
    }
    public Duplicate dup() {

        . . . . . .

        . . . . . .
    }
}
```

Vervollständigen Sie diese Klasse.

Manchmal möchte man aus einer Zahlenfolge eine neue erzeugen, bei der die ersten $n$ Folgeglieder gelöscht sind, also eine Art *Shift*-Operation. Vervollständigen Sie hierfür die Klasse *Drop*, mit der diese Operation realisiert werden kann. Für die Folge $0, 1, 2, 3, ...$ und $n = 1$ ergibt sich dann die Folge $1, 2, 3, ....$

```
public
class Drop implements Sequence {
    private Sequence s;

    ..........................................

    public Drop(int n1, Sequence s1) {
        s = s1;

        ..........................................

        ..........................................
    }

    public double next() {
        return
            s.next();
    }
    public Duplicate dup() {

        ..........................................

        ..........................................
    }
}
```

Zahlenfolgen können durch Kombination zweier anderer Zahlenfolgen konstruiert werden, zum Beispiel durch paarweises Aufsummieren oder Subtrahieren der Folgeglieder. Vervollständigen hierzu Sie die Klasse *Sub* zum Subtrahieren zweier Folgen. Für die Folgen $1, 4, 9, 16, ...$ und $0, 1, 4, 9, ...$ ergibt sich die Resultatfolge $1, 3, 5, 7, ...$

```
public class Sub implements Sequence {
    private final Sequence s1, s2;

    public Sub(Sequence s1, Sequence s2) {
        this.s1 = s1;
        this.s2 = s2;
    }
    public double next() {

        . . . . . .

        . . . . . .
    }
    public Duplicate dup() {

        . . . . . .

        . . . . . .

        . . . . . .
    }
}
```

Im folgenden wird davon ausgegeangen, dass es für alle vier Grundrechenarten die analogen Klassen zu *Sub* gibt (*Add*, *Mult*, *Sub*, *Divide*).

Eine Klasse, die wenig sinnvoll erscheint, ist die folgende:

```
public class Id implements Sequence {
    protected Sequence s;

    public Id(Sequence s1) {
        s = s1;
    }
    public double next() {
        return
            s.next();
    }
    public Duplicate dup() {
        return
            s.dup();
    }
}
```

Gibt es außer Zeilenschinderei und Rechenzeitverschwendung noch einen Grund, diese Klasse zu realisieren?

ja ☐ nein ☐

Begründung:

. . . . . .

. . . . . .

In der Mathematik wird, um das Wachstum einer Folge zu untersuchen, häufig die Differenzenfolge gebildet. Im Beispiel über die *Sub*-Klasse ist so eine Folge berechnet worden.

Vervollständigen Sie die Klasse *Diff*, mit der zu einer gegebenen Folge die Differenzenfolge berechnet werden kann.

```
public
class Diff

    . . . . . .
{

    . . . . . .

    . . . . . .

    . . . . . .

    . . . . . .

    . . . . . .

    . . . . . .

    . . . . . .

    . . . . . .

    . . . . . .

    . . . . . .

    . . . . . .

    . . . . . .
}
```

Die inverse Operation zur Bildung der Differenzenfolge ist das Aufsummieren einer Folge, zu einer Folge also die zugehörige Reihe zu berechnen. Zu einer Folge $1, 2, 4, 7, ...$ wird das Resultat $0, 1, 3, 7, 14, ...$ geliefert. Vervollständigen Sie auch diese Klasse *Sum*:

```
public class Sum implements Sequence {
    private Sequence s;

        ..........

    public Sum(Sequence s) {
        this.s = s;

        ..........

        ..........
    }
    public double next() {

        ..........

        ..........

        ..........
    }
    public Duplicate dup() {

        ..........

        ..........
    }
}
```

Als einfache Anwendung dieses Baukastens konstruieren Sie bitte eine Klasse *AritmMean*, die alleine durch die Nutzung der vorhandenen Klassen und ohne Neuimplementierung von *next* realisiert ist, und mit der die Folge der Mittelwerte der ersten $n$ Folgeglieder eine Folge berechnet werden kann. Wenn die Ausgangszahlenfolge mit den Werten $1.0, 2.0, 3.0, 4.0, 10.0, ...$ beginnt, so soll die Folge $1.0, 1.5, 2.0, 2.5, 4.0, ...$ entstehen.

**public class** ArithmMean

. . . . . .
{

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .
}

---