Prof. Dr. Uwe Schmidt

fhwedel
UNIVERSITY OF APPLIED SCIENCES

15. August 2013

Aufgaben zur Klausur **Algorithmen und Datenstrukturen in C** im SS 2013 (BECom 17b, BInf 17b, BMInf 17b, BTinf 17b, BWinf 17b, BInf 201, BTInf 201, BMInf 201,BWInf 201)

Zeit: 90 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.

Nutzen Sie die Rückseiten der Klausur zur Entwicklung der Lösungen und übertragen die fertigen Lösungen in das Aufgabenblatt.

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 9 Seiten.

Aufgabe 1:

Seien f_i, g_i Funktionen vom Typ $\mathbb{N} \to \mathbb{R}$.
Weiter gelte $f_i(n) \in O(g_i(n))$ und $c_i \in \mathbb{R}$ für $i \in \{1, 2, \dots\}$.

1. Aus welcher Komplexitätsklasse ist $f(n) = f_1(n) + c_1 * (n^2 + f_2(n))$

   .....

2. Aus welcher Komplexitätsklasse ist $f(n) = f_1(n) * f_2(n) + f_3(n)$

   .....

3. Aus welcher Komplexitätsklasse ist $f(n) = c_5 * n^5 + c_4 * n^4 + c_3 * n^3 + 100 * log_2\ n$

   .....

4. Aus welcher Komplexitätsklasse ist $f(n) = (log_2\ n + log_{10}\ n) * n$

   .....

Aufgabe 2:

Gegeben sind die folgenden Datentypdefinitionen und Prototypen

```
#include <string.h>
#include <assert.h>

typedef char * Element;

typedef struct Node * List;
struct Node {
   List next;
   Element e;
};

extern int invList(List l);
extern List cons(Element e, List l);
extern List insert(Element e, List l);
```

Entwickeln die Testfunktion *invList*, mit der überprüft werden kann, ob die in einer Liste enthaltenen Elemente in aufsteigender Reihenfolge gespeichert sind. Es soll dabei ausgeschlossen werden, dass Elemente doppelt in der Liste enthalten sind.

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Entwickeln Sie die Funktion *insert*, die ein Element in eine sortierte Liste einfügt. Die Funktion soll rekursiv arbeiten. Sie soll so arbeiten, dass die Invariante erhalten bleibt, d.h. wenn vor dem Einfügen in eine Liste l die Invariante gilt, so auch nach dem Einfügen.

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Die Funktion *cons*, die ein Element vorne an eine Liste anhängt, kann bei der Implementierung verwendet werden.

Aufgabe 3:

Vorrang–Warteschlangen werden zur Speicherung beliebig vieler Elemente oder Schlüssel–Wert–Paare verwendet, wobei nur der schnelle Zugriff auf das kleinste Element aus einer Menge gefordert wird. Das effiziente Suchen eines beliebigen Elements ist nicht gefordert. Elemente dürfen hierbei durchaus doppelt vorkommen.

Dieses kann mit einem binären Baum effizient implementiert werden, wenn der Baum so aufgebaut wird, dass die Wurzel eines jeden Teibaums immer einen Wert enthält, der nicht größer ist als die Werte, die in den Teilbäumen gespeichert werden.

Das folgende Programmstück definiert die Datentypen und ein Prädikat für die Invariante.

```
typedef double Element;

typedef struct Node * BinaryHeap;

struct Node
{
  Element info;
  BinaryHeap l;
  BinaryHeap r;
};

int isEmptyBinaryHeap(BinaryHeap h) {
  return h == (BinaryHeap)0;
}

static
int
greaterOrEqual (BinaryHeap h, Element e);

extern
int invBinaryHeap(BinaryHeap h);
```

Entwickeln Sie die Invariante einschließlich der Hilfsfunktion:

greaterOrEqual

. . . . . .

. . . . . .

. . . . . .

. . . . . .

invBinaryHeap

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Mit welcher Zeitkomplexität arbeitet die Invariante?

. . . . . .

Aufgabe 4:

Gegeben seien die folgenden Typdefinitionen und Variablendeklarationen:

```
typedef char *Key;
typedef struct ANode * Attr;
typedef struct Node *List;

struct Node
{
  Key k;
  Attr v;
  List next;
};

struct ANode
{
  char * name;
  unsigned long age[3];
};

typedef struct hashtable *Map;

struct hashtable
{
  unsigned int size;
  unsigned int card;
  List *table;
};

typedef int (* Cmp)(Key k1, Key k2);

extern int compare(Key k1, Key k2);

extern unsigned int hash(Key e);

extern Attr search(Key k, Map m, Cmp c);

extern Map mkEmpty(void);

extern Map m;

extern Key k1,k2;
```

Bestimmen Sie für die folgenden Ausdrücke den Typ gemäß ANSI-C. Nutzen Sie hiefür, wenn möglich, die deklarierten Typnamen. Sollten Ausdrücke vorkommen, die zur Übersetzungszeit Fehlermeldungen erzeugen, so kennzeichnen Sie diese mit dem Wort `FEHLER`

$*$m .....

($*$m).card == 0 .....

mkEmpty()$->$table .....

m$->$table[3] .....

$*$(m$->$table[0]$->$v) .....

search(k1,m,compare) .....

$*$(($*$(m$->$table))$->$next) .....

search(k1,m,compare(k1,k2)) .....

compare .....

m$->$table[compare(k1,k2)] .....

m$->$table[0]$->$next$->$next$->$v$->$name[2] .....

$*$(m$->$table[1]$->$next$->$next$->$v$->$age + 2) .....
