

Aufgaben zur Klausur C im SS 2013 (IA 302)

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 8 Seiten.

---

## Aufgabe 1:

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.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....



## Aufgabe 2:

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 Teilbaums 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 3:

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 hierfür, 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) .....
-