Prof. Dr. Uwe Schmidt

ptlwedel
PRIVATE BERUFSFACHSCHULE

30. Januar 2012

Aufgaben zur Klausur **C** im WS 2011/12 (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.

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

Tipp: Bei der Entwicklung der Lösung können kleine Skizzen manchmal hilfreich sein.

Viel Erfolg!

Diese Klausur besteht einschließlich dieses Deckblattes aus 9 Seiten.

Aufgabe 1:

Gegeben seien die folgenden C–Typdefinitionen und Funktionsköpfe:

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

typedef char * Element;

typedef struct node * BinTree;
struct node {
    Element info;
    BinTree l;
    BinTree r;
};

extern BinTree mkEmpty(void);
extern BinTree mkOne(Element s);
extern BinTree insert(Element s, BinTree t);

extern int isEmpty(BinTree t);
extern int isIn(Element s, BinTree t);

extern int invBinSearchTree(BinTree t);
extern int allLessThan(Element x, BinTree t);
extern int allGreaterThan(Element x, BinTree t);

extern int invBinHeap(BinTree t);
extern int isGreaterOrEqual(Element x, BinTree t);

/* works like strcmp, but result is one of -1, 0, +1 */
extern compare(Element x1, Element x2);
```

Nutzen Sie die in dem Programmstück vorgegebenen Datentypen und Funktionsdeklarationen für die zu entwickelnden Funktionen.

Mit dem Datentyp *BinTree* können sowohl binäre Suchbäume als auch binären Halden realisiert werden.

Implementieren Sie die Funktion *invBinSearchTree* für einen Test, ob ein Baum einen binären Suchbaum repräsentiert. Modularisieren Sie diese Funktion so, dass Teile des Tests von den Funktionen *allLessThan* und *allGreaterThan* übernommen werden.

Die Funktion *invBinSearchTree*:

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Die Funktion *allLessThan*:

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Binäre Halden arbeiten mit der gleichen Struktur, nur hat die Invariante *invBinHeap* eine andere Gestalt. Modularisieren Sie diese Funktion so, dass Teile des Tests von der Funktion *isGreaterOrEqual* übernommen werden.

Die Funktion *invBinHeap*:

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Die Funktion *isGreaterOrEqual*:

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Entwickeln Sie für binäre Suchbäume die Suchfunktion *isIn*

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Entwickeln Sie für binäre Halden die Suchfunktion *isIn*

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

In welcher Zeitkomplexität in Bezug auf die Anzahl der Elemente in der Halde läuft diese Funktion?

. . . . . .

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 eines 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*:

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .
