Prof. Dr. Uwe Schmidt

fhwedel
UNIVERSITY OF APPLIED SCIENCES

16.August 2011

Aufgaben zur Klausur **Algorithmen und Datenstrukturen in C** im SS 2011 (BInf 201, BTInf 201, BMInf 201, BWInf 201)

Zeit: 75 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 9 Seiten.

Aufgabe 1:

Gegeben seien die folgenden Datentypdefinitionen und Funktionsdeklarationen für die Verarbeitung von Matrizen:

```
typedef double Element;

typedef Element *Row;
typedef Row *Rows;

typedef struct
{
  Rows rows;
  Element *elems;
  int width;
  int height;
} *Matrix;

/* constructor functions */

extern Matrix newMatrix (int w, int h);
extern Matrix zeroMatrix (int w, int h);
extern Matrix unitMatrix (int w, int h);

/* destructor */

extern void freeMatrix (Matrix m);

/* space statistics */

extern double sizePerElement (Matrix m);

/* matrix ops */

extern Matrix addMatrix (Matrix m1, Matrix m2);
extern Matrix transposeMatrix (Matrix m);

/* element access ops */

extern Element at (Matrix m, int i, int j);
extern Matrix setAt (Matrix m, int i, int j, Element v);
```

Tipp: Zum Verständnis dieser Datenstruktur und der Konstruktion von Matrizen (*newMatrix* auf der folgenden Seite) kann eine Skizze nützlich sein.

Die *newMatrix*–Funktion sei wie folgt implementiert:

```
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>

/*————————————*/

Matrix
newMatrix (int w, int h)
{
  Matrix res = malloc (sizeof (*res));
  if (res)
    {
      res->rows = malloc (h * sizeof (Row));
      res->elems = malloc (h * w * sizeof (Element));
      res->width = w;
      res->height = h;
      {
        Rows rs = res->rows;
        Row r = res->elems;
        if (rs && r)
          {
            while (h--)
              {
                *rs++ = r;
                r += w;
              }
            return res;
          }
      }
    }
  /* heap overflow */
  perror ("newMatrix:  can't allocate matix");
  exit (1);
}
```

Für die Berechnung der Speicherplatzeffizienz dieser Matrizenimplementierung gibt es die Funktion *sizePerValue*. Diese Funktion soll berechnen, um welchen Faktor sich der benötigte Platz pro Element bei dieser Datenstruktur erhöht, es muss also der Quotient aus dem gesamten benötigten Speicher und dem Platz pro Element mal Anzahl der Elemente berechnet werden.

Die Funktion *sizePerElement*:

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Implementieren Sie die Routinen *at* und *setAt* für den lesenden und schreibenden indizierten Zugriff, und zwar so, dass eine Indexüberprüfung mit *assert* vorgenommen wird.

Die Funtion *at*:

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Die Funktion *setAt* soll einen Wert *v* an einer Stelle in der Matrix setzen und die veränderte Matrix als Wert zurückgeben. Die Funktion *setAt*:

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Aufgabe 2:

Gegeben sei das folgende (unvollständige) C–Programmstück für die Implementierung von binären Suchbäumen. Alle Programteile, die zur Lösung der Aufgabe nicht notwendig sind, sind hier weggelassen.

```
typedef int Element;

int compare(Element e1, Element e2) {
   return (e1 >= e2) − (e1 <= e2);
}

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

#define isEmpty(b) ((b) == 0)

int searchMin(Element e, BinTree t, Element ∗ min);
```

Entwickeln Sie die Routine **searchMin**. Diese Funktion soll das kleinste Element in dem Baum suchen, das größer oder gleich dem Parameter e ist. Sie soll als Funktionsresultat berechnen, ob ein solches Element existiert. Im Parameter min soll im Fall der Existenz der gesuchte Wert zurückgegeben werden.

```
int searchMin (Element e, BinTree t, Element * min)
{
  if (isEmpty (t))
     ..........................................................
     ..........................................................
  switch (compare (e, t->info))
    {
    case -1:
     ..........................................................
     ..........................................................
     ..........................................................
     ..........................................................
    case 0:
     ..........................................................
     ..........................................................
     ..........................................................
     ..........................................................
    case +1:
     ..........................................................
     ..........................................................
     ..........................................................
     ..........................................................
    }
}
```

Aufgabe 3:

Gegeben seien die folgenden Typdefinitionen und Variablendeklarationen:

```
#include <stdlib.h>

typedef struct X * Tree;

typedef Tree (*Tf)(Tree);

struct X {
  char * k;
  struct D * a;
  Tree cs[2];
};

struct D {
  Tf f;
  char * n;
  unsigned int i;
};

Tree root;

long int li = 2;
```

Bestimmen Sie für die folgenden Ausdrücke den Typ gemäß ANSI-C. Vorsicht: Sollten Ausdrücke vorkommen, die zur Übersetzungszeit Fehlermeldungen erzeugen, so kennzeichnen Sie diese mit dem Wort `FEHLER`

root→k .....

root→k[1] .....

∗((∗root).k) .....

(∗(root→cs))→a→f .....

root→a→i + li .....

(root→a→f)(root) .....

root→a→i++ .....

∗(root→cs + 1) .....

root→cs[li] .....

root ? root→cs : 0 .....

root ? root→cs[0] : root .....

sizeof ∗root .....

sizeof (**struct** X) .....
