Prof. Dr. Uwe Schmidt

fhwedel
UNIVERSITY OF APPLIED SCIENCES

18.August 2009

Aufgaben zur Klausur **Algorithmen und Datenstrukturen in C** im SS 2009 (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 sei das folgende C–Programm zur Verarbeitung von Mengen als Bitstrings.

```
#include <stdio.h>

typedef unsigned char Set;
#define SetMax 8

void print(Set s) {
  char * digits = "0123456789abcdef";
  printf("%c%c\n",
         (char)(digits[(s>>4) & 0x0f]),
         (char)(digits[s & 0x0f]));
}

#define single(i) ( (Set)(1 << (i)) )
#define first(n) (single(n) - 1)
#define interval(n,m) (first(m+1) ^ first(n))

int main(void) {
    Set s1;

    print( (Set)-1 );
    print( single(2) );
    print( single(SetMax / 2 + 1) );

    print( interval(2,5) );
    print( interval(5,2) );
    print( interval(1,SetMax) );

    print( 5 | 10 );
    print( 5 || 10 );
    print( first(SetMax / 2 - 1) );

    s1 = interval(2,4) & ~interval(3,5); print(s1);
    s1 = interval(2,4) ^ interval(3,5); print(s1);

    s1 = 4 + 16 + 64;
    s1 ^= s1 & (~s1 + 1); print(s1);

    return 0;
}
```

Die Mengen sind in diesem Beispiel 8 Bits lang, können also die Elemente $0, 1, \ldots, 7$ enthalten. *print* gibt eine Menge als 2–stellige Hexadezimalzahl aus. Die Menge, die die Elemente 1,2 und 3 enthält, würde als $0e$ ausgegeben werden.

Welche 12 Ausgabezeilen erzeugt dieses Programm?

1) .....

2) .....

3) .....

4) .....

5) .....

6) .....

7) .....

8) .....

9) .....

10) .....

11) .....

12) .....

Aufgabe 2:

Seien f_1, f_2, g_1, g_2 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 $i \in \{1, 2\}$.

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

   .....

2. Aus welcher Komplexitätsklasse ist $f(n) = (f_1(n))^2 + f_1(n)$

   .....

3. Aus welcher Komplexitätsklasse ist $f(n) = f_1(n) * (f_2(n) + c_2 * f_1(n))$

   .....

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

   .....

Aufgabe 3:

Gegeben seien die C Makro-, Typ-, Variablen- und Funktionsdeklarationen und -definitionen:

```
typedef double Element;

typedef Element *Row;
typedef Row *Rows;

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

#define trunc(x) ((int)(x))
#define at(a,i,j) ((a)->rows[i][j])

typedef struct Descr * Matrix;

typedef Matrix (* MatrixConstr)(int w, int h);
typedef Matrix (* UnaryMatrixOp)(Matrix m);

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

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

Matrix m1;
MatrixConstr c1;
```

Bestimmen Sie für die folgenden Ausdrücke den Typ gemäß ANSI-C. Sollten fehlerhafte oder logisch nicht sinnvolle Ausdrücke vorkommen, so kennzeichnen Sie diese mit dem Wort FEHLER. Nutzen Sie für die Typbestimmung die in dem Programmstück deklarierten Typnamen.

m1->elems .....

*((*m1).elems) .....

m1->rows[1][2] .....

m1->rows[m1->width] .....

m1->rows[1,2] .....

at(m1,0,0) .....

at(m1,trunc(at(m1,1,1)),+1) .....

zeroMatrix .....

c1(1,1) .....

at(c1(1,1),1,1) .....

c1 = unitMatrix .....

c1 = transposeMatrix .....

(*addMatrix(m1,m1)).rows + 1 .....

Aufgabe 4:

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.
