Prof. Dr. Uwe Schmidt

ptlwedel
PHYSIKALISCH-TECHNISCHE LEHRANSTALT

1. Februar 2006

Aufgaben zur Klausur **C** im WS 2005/06 (IA 302)

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 8 Seiten

---

Aufgabe 1:

Seien f_1, f_2, g_1, g_2 Funktionen vom Typ $\mathbb{N} \to \mathbb{R}$.
Weiter gelte $f_1(n) \in O(g_1(n))$, $f_2(n) \in O(g_2(n))$ und $c_i \in \mathbb{R}$ für $i \in \{0, 1, 2, 3\}$.

1. Aus welcher Komplexitätsklasse ist $f(n) = f_1(n) + f_2(n)$

   .....

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

   .....

3. Aus welcher Komplexitätsklasse ist $f(n) = c_1 * f_1(n)$

   .....

4. Aus welcher Komplexitätsklasse ist $f(n) = c_3 * n^3 + c_2 * n^2 + c_1 * n + c_0$

   .....

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:

    .....

    .....

    .....

    .....
    }
}
```

Sei n die Anzahl der Elemente, die in einem Baum gespeichert sind.

1. Mit welcher Zeitkomplexität arbeitet diese Funktion im Mittel?

   .....

2. Mit welcher Zeitkomplexität arbeitet diese Funktion im schlechtesten Fall?

   .....

3. Mit welcher Zeitkomplexität würde diese Funktion arbeiten, wenn eine unsortierte verkettete Liste zur Speicherung der Elemente verwendet werden würde?

   .....

4. Mit welcher Zeitkomplexität würde diese Funktion arbeiten, wenn eine sortierte verkettete Liste zur Speicherung der Elemente verwendet werden würde?

   .....

5. Mit welcher Zeitkomplexität würde diese Funktion arbeiten, wenn eine binäre Halde zur Speicherung der Elemente verwendet werden würde?

   .....

Aufgabe 3:

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

```
typedef double (*F) (double x, double y);

typedef struct x *X;

struct x
{
  int i;
  char c;
  X l[2];
  X *p;
  unsigned int u;
  F f;
};

X x1;
long int i;
double f1 (double x, double y);
double f2 (double x, double y);
double f3 (int x, int y);
```

und die folgenden C–Ausdrücke

1. x1->p
2. &x1
3. x1->f = f1
4. i ? f1 : f2
5. f1 == f2
6. f1(1,2) == f3(1,2)
7. x->f = f3
8. *(x1->p)
9. *(x1->p[0])
10. *(x1->l[2])
11. i + x1->u
12. x1->u & x1->i

1. Welche Ausdrücke sind fehlerhafte C–Ausdrücke oder enthalten logische Fehler? (Diese Ausdrücke sind in den folgenden Fragen nicht mehr zu berücksichtigen).

   1. ☐ 2. ☐ 3. ☐ 4. ☐ 5. ☐ 6. ☐ 7. ☐ 8. ☐ 9. ☐ 10. ☐ 11. ☐ 12. ☐

2. Welche Ausdrücke besitzen einen Typ *Zeiger auf Zeiger auf ...*?

   1. ☐ 2. ☐ 3. ☐ 4. ☐ 5. ☐ 6. ☐ 7. ☐ 8. ☐ 9. ☐ 10. ☐ 11. ☐ 12. ☐

3. Welche Ausdrücke besitzen einen vorzeichenlosen ganzzahligen Typ?

   1. ☐ 2. ☐ 3. ☐ 4. ☐ 5. ☐ 6. ☐ 7. ☐ 8. ☐ 9. ☐ 10. ☐ 11. ☐ 12. ☐

4. Welche Ausdrücke werden bei beliebiger Variablenbelegung immer zu 0 oder 1 ausgewertet?

   1. ☐ 2. ☐ 3. ☐ 4. ☐ 5. ☐ 6. ☐ 7. ☐ 8. ☐ 9. ☐ 10. ☐ 11. ☐ 12. ☐

5. Welche Ausdrücke besitzen einen Funktionszeiger als Typ?

   1. ☐ 2. ☐ 3. ☐ 4. ☐ 5. ☐ 6. ☐ 7. ☐ 8. ☐ 9. ☐ 10. ☐ 11. ☐ 12. ☐

6. Welche Ausdrücke besitzen einen *struct x*–Typ?

   1. ☐ 2. ☐ 3. ☐ 4. ☐ 5. ☐ 6. ☐ 7. ☐ 8. ☐ 9. ☐ 10. ☐ 11. ☐ 12. ☐

Aufgabe 4:

Das folgende Programmstück definiert die Datentypen für die Implementierung einer Liste als einfach verkettetem Ring.

```
typedef char * Element;

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

extern Ring concat(Ring r1, Ring r2);
```

Entwickeln Sie die fehlende *concat*–Routine zur Konkatenation zweier Listen.

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Mit welcher Zeitkomplexität arbeitet diese Routine, wenn n_1 und n_2 die Längen der beiden Listen bezeichnen.

. . . . . .