Prof. Dr. Uwe Schmidt

ptlwedel
PHYSIKALISCH-TECHNISCHE LEHRANSTALT

8. Februar 2007

Aufgaben zur Klausur **C** im WS 2006/07 (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 C–Programmstück für die Berechnung der Länge der kürzeszen und längsten Pfade in binären Bäumen:

```
typedef unsigned int Nat;

typedef char * Element;

typedef struct Node *Set;
struct Node {
  Element info;
  Set l;
  Set r;
};

#define isEmpty(s) ((s) == (Set)0)

Nat max(Nat i, Nat j) {
  return i > j ? i : j;
}

Nat min(Nat i, Nat j) {
  return i < j ? i : j;
}

Nat maxPathLength(Set s) {
  if (isEmpty(s))
    return 0;
  return
    1 + max(maxPathLength(s->l),
            maxPathLength(s->r));
}

Nat minPathLength(Set s) {
  if (isEmpty(s))
    return 0;
  return
    1 + min(minPathLength(s->l),
            minPathLength(s->r));
}
```

Beim Testen, ob ein Baum ausgewogen ist, benötigt man sowohl die Länge des kürzesten als auch des längsten Pfades. Dafür muss man den Baum sowohl mit *minPathLength* als auch mit *maxPathLength* traversieren.

Effizienter ist es aber, beim Traversieren sowohl die minimale als auch die maximale Pfadlänge gleichzeitig zu berechnen. Damit tritt das Problem auf, dass eine Funktion zwei Resultate liefert. Dieses kann man in C auf zwei Arten lösen, einmal durch einen Record als Resultattyp, zum Anderen mit Referenzparametern. Der zweite Weg soll hier eingeschlagen werden. Aus Symmetriegründen sollen hier beide Resultate als Referenzparameter realisiert werden. Die Funktion wird syntaktisch also zu einer Prozedur.

Vervollständigen Sie das folgende Codesück entsprechend:

Globale Variablen:

Nat minPath, maxPath;
Set s;

Ein möglicher Aufruf:

... minMaxPath(s, &minPath, &maxPath); ...

Die zu vervollständigende Prozedur:

void minMaxPath(Set s, ... minp, ... maxp) {

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

}

Aufgabe 3:

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

```
typedef double Time;

typedef Time (*F) (void);

typedef struct x *X;

struct x
{
  int x;
  unsigned char c[5];
  unsigned int n;
  Time t;
  X l[2];
  X *p;
  F getTime;
};

X x1;
long int i;
double f1 (void);
double f2 (double x);
```

und die folgenden C–Ausdrücke

1. x1->l
2. &(x1->p)
3. x1->getTime == f1
4. (x1->getTime) = f2
5. (x1->getTime)() = f2(1.0)
6. *(x1->l)
7. *(x1->l[1])
8. x1->c
9. *((*x1).c)
10. x1 && i
11. *(x1->c) & i
12. i ? x1->x : x1->n

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 ...*?

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

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

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

4. Welche Ausdrücke besitzen einen vorzeichenlosen ganzzahligen Typ?

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

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

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

6. Welche Ausdrücke besitzen einen Funktionszeiger als Typ?

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

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

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

Aufgabe 4:

Gegeben sei das folgende C–Programm zur Verarbeitung von Mengen als Bitstrings.

```
#include <stdio.h>

typedef unsigned char Set;
#define SetMax 8

void printSet(Set s) {
    unsigned int i = SetMax;
    while ( i-- != 0 )
        printf("%1u", (unsigned int)((s >> i) & 1));
}

#define PRINT(s) { printSet(s); printf("\n"); }

#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(0) );
    PRINT( single(SetMax / 2 - 1) );

    PRINT( interval(2,6) );
    PRINT( interval(6,2) );
    PRINT( interval(0,SetMax) );

    PRINT( 4 | 15 );
    PRINT( 4 || 15 );
    PRINT( first(SetMax / 2 + 1) );

    s1 = interval(2,5) & ~interval(3,6); PRINT(s1);
    s1 = interval(2,5) ^ interval(3,6); PRINT(s1);

    s1 = 2 + 8 + 32;
    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. *printSet* gibt eine Menge im Binärformat aus. Die Menge, die nur die 1 enthält würde als 00000010 ausgegeben werden. Das *PRINT* Makro gibt jeweils eine Menge pro Zeile aus.

Welche 12 Ausgabezeilen erzeugt dieses Programm?

1) .....

2) .....

3) .....

4) .....

5) .....

6) .....

7) .....

8) .....

9) .....

10) .....

11) .....

12) .....