Prof. Dr. Uwe Schmidt

ptlwedel
PHYSIKALISCH-TECHNISCHE LEHRANSTALT

5. Februar 2008

Aufgaben zur Klausur **C** im WS 2007/08 (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 11 Seiten

---

Aufgabe 1:

Rot–Schwarz–Bäume sind binäre Suchbäume, die gewisse Ausgewogenheitskriterien erfüllen müssen. Diese Bedingungen werden mit Hilfe Invarianten formuliert. Eine Bedingung ist die, dass keine roten Knoten einen roten Kindknoten besitzen.

Die Datenstruktur–Definition für einen als Rot–Schwarz–Baum realisierten Mengendatentyp habe folgendes Aussehen:

```
typedef int Element;

typedef struct Node * Set;

struct Node
{
  enum { RED, BLACK } color;
  Element info;
  Set l;
  Set r;
};

static struct Node finalNode = {BLACK, 0, 0, 0};

#define mkEmptySet() (&finalNode)
#define isEmptySet(s) ((s) == &finalNode)

extern unsigned int maxPathLength(Set s);
extern unsigned int minPathLength(Set s);

#define isBlackNode(s) ((s)->color == BLACK)
#define isRedNode(s) (! isBlackNode(s))

extern int hasRedChild(Set s);
extern int invNoRedNodeHasRedChild(Set s);
```

In diesem Codefragment sind einige Makros und einige Funktionen deklariert. Die Bedeutung der Funktionen geht aus dem Namen hervor. Benutzen Sie bitte diese Makros und Funktionen zur Formulierung Ihrer Lösung.

Man erkennt an den Makros *isEmptySet* und *mkEmptySet*, dass in dieser Implementierung der leere Baum durch einen Zeiger auf einen speziellen Knoten repräsentiert wird, nicht durch den 0–Zeiger.

Es soll als erstes die Hilfsfunktion *hasRedChild* entwickelt werden. Dieses Prädikat soll überprüfen, ob für einen beliebigen Baum ein möglicher Wurzelknoten einen roten Kindknoten besitzt.

Die Funktion *hasRedChild*:

Mit dieser Hilfsfunktion kann der Teil der Invariante für Rot–Schwarz–Bäume formuliert werden, der überprüft, dass an keiner Stelle in einem Baum ein roter Knoten einen roten Kindknoten besitzt. Dieses überprüft die Funktion *invNoRedNodeHasRedChild*

Die Funktion *invNoRedNodeHasRedChild*:

Aufgabe 2:

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→cs[li] .....

∗((∗root).k) .....

(∗(root→cs))→a→f .....

∗(root→cs + 1) .....

root→a→i + li .....

(root→a→f)(root) .....

root→a→i++ .....

root ? root→cs : 0 .....

root ? root→cs[0] : root .....

sizeof ∗root .....

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

Des weiteren seien folgende Funktionen definiert:

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

#include "types.c"

static unsigned int cnt = 0;

Tree mkTree(char * k,
            struct D * a,
            Tree t1,
            Tree t2) {
  Tree res = malloc(sizeof *res);
  if (! res) exit(1);
  res->k = k;
  res->a = a;
  res->cs[0] = t1;
  res->cs[1] = t2;
  return res;
}

struct D * mkD(Tf f, char * n) {
  struct D * res = malloc(sizeof *res);
  if (! res) exit(1);
  res->f = f;
  res->n = n;
  res->i = cnt++;
  return res;
}

Tree left(Tree t) {
  assert(t != 0);
  return t->cs[0];
}

Tree right(Tree t) {
  assert(t != 0);
  return t->cs[1];
}
```

Welche Ausgaben erzeugt das folgende Programm? Es können in diesem Programm Ausdrücke vorkommen, die nicht definiert sind. Kennzeichnen Sie diese mit UNDEF in der Lösungszeile. Geben Sie alle Ausgaben an, auch wenn vorher undefinierte Auswertungen gemacht wurden.

```
#include <stdio.h>

#include "mk.c"

int main(void) {
  Tree t1, t2;
  struct D *f1, *f2;
  f1 = mkD(left,"links");
  f2 = mkD(right,"rechts");

  t1 = mkTree("zwiebel", f1, 0, 0);
  t2 = mkTree("moere", f2, t1, 0);
  root = mkTree("wurzel", f2, t1, t2);

  printf("1) %s\n", root->k+1);
  printf("2) %d\n", (int)(root->a->i));
  printf("3) %s\n", root->a->n + 1);
  printf("4) %s\n", (root->a->f)(root)->k);
  printf("5) %s\n", root->cs[0]->a->n);
  printf("6) %s\n", (*(root->cs[1]->cs))->k);
  printf("7) %d\n", (*(root->cs))->cs[0]->a->i);

  return 0;
}
```

1) .....

2) .....

3) .....

4) .....

5) .....

6) .....

7) .....

Aufgabe 3:

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, s2;

    s1 = 0xda; PRINT(s1);
    s2 = -s1; PRINT(s2);
    s2 = s1 & -s1; PRINT(s2);
    s2 = s1 & (~s1 + 1); PRINT(s2);
    s2 = s1 && (~s1 - 2); PRINT(s2);
    s2 = s1 ^ -s1; PRINT(s2);
    s2 = s1 ^ (s1 & (~s1 + 1)); PRINT(s2);
    s2 = (s1 ^ s1) & (~s1 + 1); PRINT(s2);
    s2 = (s1 | first(4)) + 1; PRINT(s2);
    s2 = s1 | single(5); PRINT(s2);

    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 10 Ausgabezeilen erzeugt dieses Programm unter der Annahme, dass die Maschine mit 2er-Komplement Zahlendarstellung arbeitet?

1) .....

2) .....

3) .....

4) .....

5) .....

6) .....

7) .....

8) .....

9) .....

10) .....
