Prof. Dr. Uwe Schmidt

fhwedel
UNIVERSITY OF APPLIED SCIENCES

3. Februar 2009

Aufgaben zur Klausur **C** und **Objektorientierte Programmierung** im WS 2008/09 (WI h103, II h105, MI h353)

Zeit: 150 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 18 Seiten.

Aufgabe 1:

Gegeben sei der folgende Ausschnitt eines C–Moduls für die Implementierung von binären Suchbäumen.

```
typedef long int Element;

#define compare(x,y) (((x) > (y)) - ((x) < (y)))

typedef struct Node *Set;

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

Set mkEmptySet(void);
int isEmptySet(Set s);

Set mkOneElemSet(Element e);
Set insertElem(Element e, Set s);
Set removeElem(Element e, Set s);


Set
insertElem(Element e, Set s)
{
   if (isEmptySet(s))
      return mkOneElemSet(e);

   switch (compare(e, s->info))
      {
      case -1:
         s->l = insertElem(e, s->l);
         break;
      case 0:
         break;
      case +1:
         s->r = insertElem(e, s->r);
         break;
      }

   return s;
}
```

```
static
Set removeRoot(Set s) {
  ...
}

Set
removeElem(Element e, Set s)
{
  if (isEmptySet(s))
    return s;

  switch (compare(e, s->info))
    {
    case -1:
      s->l = removeElem(e, s->l);
      break;
    case 0:
      s = removeRoot(s);
      break;
    case +1:
      s->r = removeElem(e, s->r);
      break;
    }

  return s;
}


Set changeElem(Element e, Set s, ...) {
  ...
}

Set
removeElem1(Element e, Set s) {
  return
    changeElem(e, s, ...);
}

Set
insertElem1(Element e, Set s) {
  return
    changeElem(e, s, ...);
}
```

Man erkennt, dass die Funktionen für das Einfügen und Löschen sehr ähnlich sind. Diese Funktionen kann man zusammenfassen zu einer Funktion *changeElem*, die die Unterschiede der beiden Funktionen durch zusätzliche Parameter geliefert bekommt.

Entwickeln Sie die Funktion *changeElem*.

Definieren sie zuerst die Typen der zusätzlichen Parameter.
Typdefinitionen:

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Der Programmcode für die Funktion *changeElem*.

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Der Programmcode für die neue Funktion zum Einfügen *insertElem1* und erforderliche Hilfkonstrukte:

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Der Programmcode für die neue Funktion zum Löschen *removeElem1* und erforderliche Hilfkonstrukte:

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Aufgabe 2:

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

```
typedef double Random;

typedef Random (*F) (void);

typedef struct x *R;

struct x
{
  unsigned int n;
  Random t;
  R l[2];
  R *p;
  F getRandom;
  int x;
  unsigned char c[5];
};

R x1;
long int i;
double rf1 (void);
double rf2 (double x);
```

und die folgenden C–Ausdrücke

1. &(x1->p)
2. x1->getRandom == rf1
3. *(x1->l[1])
4. x1->l
5. *(x1->l)
6. (x1->getRandom)() = rf2(1.0)
7. x1->c
8. *((*x1).c)
9. *(x1->c) & i
10. i ? x1->x : x1->n
11. x1 && i
12. (x1->getRandom) = rf2

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 vorzeichenlosen ganzzahligen Typ?

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

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

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

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

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

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

   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 werden bei beliebiger Variablenbelegung immer zu 0 oder 1 ausgewertet?

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

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));
        if (i == 4)
          printf("  ");
    }
}
static unsigned int linecnt = 0;
#define PRINT(s) { printf("%2u)  ", ++linecnt); 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 = 0xad; PRINT(s1);
    s2 = 1-s1; PRINT(s2);
    s2 = -s1; PRINT(s2);
    s2 = ~s1 + 1; PRINT(s2);
    s2 = s1 & 0x3c; PRINT(s2);
    s2 = s1 | 0xf0; PRINT(s2);
    s2 = s1 && (s1 - 4); PRINT(s2);
    s2 = (s1 ^ s1) & (~s1 + 1); PRINT(s2);
    s2 = s1 ^ (s1 & (~s1 + 1)); PRINT(s2);
    s2 = s1 & interval(2,6); PRINT(s2);
    s2 = s1 ^ single(3); 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 $0000\ 0010$ ausgegeben werden. Das *PRINT* Makro gibt jeweils eine Menge pro Zeile aus.

Welche 11 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) .....

11) .....

Aufgabe 4:

Gegeben sei das folgende Programm:

```
#include <stdio.h>

char * tab [] = {"Haschisch", "Unterleib", "Geschenk", "Blei", "Laster"};

char * * ptab [] = { tab + 4, tab + 3, tab + 2, tab + 1, tab };

char * * * ppp = ptab;

int main ( int argc , char * argv [] )
{
   printf( "%s\n" , * (* (ppp + 3) - 1) + 6 );
   printf( "%s\n" , ppp [3] [0] + 6 );
   printf( "%s\n" , * (* (ppp + 2)) + 5 );
   printf( "%s\n" , * (* ++ppp) + 1 );
   printf( "%s\n" , * (* --ppp) + 2 );

   return 0;
}
```

Welche Ausgabezeilen liefert dieses Programm:

1) .....

2) .....

3) .....

4) .....

5) .....

Wie viel Speicher wird von den Variablen tab, ptab und ppp und den in den Initialisierungen vorkommenden Konstanten benötigt? Geben Sie hierfür einen Ausdruck mit dem **sizeof**-Operator an.

.....

.....

Aufgabe 5:

Lesen Sie bitte vor der Bearbeitung die gesamte Aufgabe durch. Es sind in der Aufgabenstellung Vorwärtsreferenzen enthalten.

Zweidimensionale geometrische Figuren (Kreise, Rechtecke, ...) können auf sehr unterschiedliche Arten in einem System implementiert werden. Eine sehr allgemeine und in manchen Anwendungen sehr speicherplatzeffiziente Art ist die, Figuren durch Funktionen zu repräsentieren, z.B. durch folgende Schnittstelle in Java:

```
interface Figure {
    public boolean contains(Point p);
}
```

Die so repräsentierten Figuren können beliebig groß sein, zum Beispiel einen ganzen Quadranten darstellen, im Extemfall sogar die gesamte Ebene. Das Visualisieren solcher Figuren kann implementiert werden, indem der gesamte darzustellende Bereich *abgetastet* wird, d.h. es wird punktweise getestet, ob eine Koordinate in einer Figur liegt oder außerhalb. Dieses wird beispielhaft in dem Testprogramm am Ende der Aufgabe gemacht.

Einige häufig verwendete Figuren sind in einer Schnittstelle gesammelt:

```
interface SimpleFigures {

  // useful simple figures


  public static final Figure unitSquare =
    new Square(Point.org,1.0);

  public static final Figure unitCircle =
    new Circle(Point.org, 1.0);

  public static final Figure halfPlaneX =
    new Figure() {
      public boolean contains(Point p) {
        return
          p.x >= 0;
      }
    };
}
```

In der Schnittstelle sind drei Figuren vordefiniert, das Einheitsquadrat, ein Kreis um den Ursprung mit dem Radius 1.0 und die Halbebene rechts der y–Achse (alle Werte mit positiver x–Koordinate).

Erweitern Sie die Schnittstelle *SimpleFigures* um eine neue Figur, *halfPlaneD*, die die abgeschlossene Halbebene oberhalb der Hauptdiagonalen repräsentiert.

public static final Figure halfPlaneD =

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Die Klasse *Circle* ist wie folgt definiert:

```
public class Circle implements Figure {

   private Point org;
   private double radius;

   public Circle(Point o, double r) {
      org = o;
      radius = r;
   }
   public boolean contains(Point p) {
      double dx = p.x − org.x;
      double dy = p.y − org.y;
      return
         Math.sqrt(dx∗dx + dy∗dy) <= radius;
   }
}
```

Entwickeln Sie analog dazu eine Klasse *Square*

```
public class Square implements Figure {

      ..........................................

      ..........................................

   public Square( ..........................................  ) {

            ..........................................

            ..........................................
   }
   public boolean contains(Point p) {

            ..........................................

            ..........................................

            ..........................................

            ..........................................
   }
}
```

Die Koordinaten, die Punkte, werden in diesem Beispiel durch folgende Klasse realisiert:

```
final
public class Point {
    final public double x;
    final public double y;

    public Point(double x1, double y1) {
        x = x1; y = y1;
    }
    static final public Point org = new Point(0.0, 0.0);
    static final public Point x1 = new Point(1.0, 0.0);
    static final public Point y1 = new Point(0.0, 1.0);
    static final public Point xy = new Point(1.0, 1.0);
}
```

In der Schnittstelle sind einige häufig verwendete Punkte vordefiniert.

Figuren können auf viele unterschiedliche Arten manipuliert werden. Ein Transformationstyp ist der, dass eine punktweise Transformation, z.B. eine Verschiebung oder Drehung, vorgenommen wird. Transformationen können ebenfalls durch Objekte repräsentiert werden. Die folgende Klasse wird in diesem Beispiel verwendet:

```
interface Transform {

  public Point move(Point p);

  // useful elementary transformations

  static final public Transform mirror =
    new Transform() {
      public Point move(Point p) {
        return
          new Point(−p.x, −p.y);
      }
    };

  static final public Transform rotate90 =
    new Transform() {
      public Point move(Point p) {
        return
          new Point(−p.y, p.x);
      }
    };
}
```

In dieser Schnittstelle sind zwei Transformationen durch die Verwendung von anonymen Klassen realisiert, eine punktweise Spiegelung am Ursprung und eine 90–Grad–Drehung.

Mit den Methoden aus der Translationsklasse können Figuren verschoben werden.

```
public class Translation implements Transform {

    private Point p;

    public Translation(Point p1) {
        p = p1;
    }

    public Point move(Point p1) {
        return
            new Point(p1.x − p.x, p1.y − p.y);
    }
}
```

Entwickeln Sie analog zur Translationsklasse eine Klasse für die Erzeugung von Skalierungen, also von Transformationen, die alle Kooordinaten mit einem festen Wert in x–Richtung scalieren, und mit einem zweiten Wert in y–Richtung.

```
public class Scaling implements Transform {

    ..........

    ..........

    public Scaling( ..........) {

        ..........

        ..........
    }

    public Point move(Point p1) {

        ..........

        ..........

        ..........

        ..........
    }
}
```

Eine Transformation kann auf eine Figur angewendet werde. Das Resultat ist wieder eine Figur. Dieser Prozess wird ebenfalls durch eine Klasse implementiert. Vervollständigen sie diese Klasse:

```
public class TransformedFigure implements Figure {

        ..........

        ..........

    public TransformedFigure( ..........) {

        ..........

        ..........

    public boolean contains(Point p) {

        ..........

        ..........

        ..........

        ..........

    }
}
```

Zwei oder mehrere Figuren können auf unterschiedliche Weise kombiniert werden, zum Beispiel sind alle Mengenoperationen (Vereinigung, Durchschnitt, Subtraktion, ...) möglich. Die zweistellige Kombination wird wieder mit Hilfe einer Klasse beschrieben:

```
abstract public class CombinedFigure implements Figure {
    protected Figure fig1;
    protected Figure fig2;

    protected CombinedFigure(Figure f1, Figure f2) {
        fig1 = f1; fig2 = f2;
    }
}
```

Entwickeln Sie eine abgeleitete Klasse für die Vereinigung zweier Figuren.

```
public class UnionFigure extends CombinedFigure {

    public UnionFigure( ..... ) {

        .....

        .....
    }

    public boolean contains(Point p) {

        .....

        .....

        .....

        .....
    }
}
```

Gegeben sei die folgende Mini–Anwendung:

```
class Main implements SimpleFigures {
   public static void main(String [] args) {
      Figure fig1 = unitCircle;
      Figure fig2 = new UnionFigure(fig1, unitSquare);
      Figure fig3 = new TransformedFigure(fig2, new Scaling(new Point(0.4,0.4)));
      Figure fig4 = new TransformedFigure(fig3, new Translation(new Point(0.5,0.5)));
      scan(fig1,10,10);
      scan(fig2,10,10);
      scan(fig3,10,10);
      scan(fig4,10,10);
   }
   static void scan(Figure fig, int w, int h) {
      for (int j = h−1; j >= 0; −−j) {
         for (int i = 0; i < w; ++i)
            scan(fig.contains(new Point((double)j/h, (double)i/w)), j, i);
         System.out.println();
      }
      System.out.println();
   }
   static void scan(boolean c, int y, int x) { System.out.print(c ? 'X' : '.'); }
}
```

Wie viele Objekte der Klasse *Point* werden bei der Ausführung von *scan(fig1,10,10);* erzeugt?

. . . . . .

Wie viele Objekte der Klasse *Point* werden bei der Ausführung von *scan(fig3,10,10);* erzeugt?

. . . . . .

Wie viele unterschiedliche Objekte sind über die Variablen *fig1*,*fig2*,*fig3* und *fig4* referenzierbar?

. . . . . .

Wie viele Objekte der Klasse *TransformedFigure* oder Unterklassen dieser Klasse werden bei der Ausführung des gesamten Programms erzeugt?

. . . . . .