Fachhochschule Wedel
Prof. Dr. Uwe Schmidt

15. August 2000

Aufgaben zur Klausur **C** und **Objektorientierte Programmierung** im SS 2000 (WI h103, II h105, MI h353)

Zeit: 120 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 14 Seiten

---

Aufgabe 1:

Die folgenden Klassen dienen zur Implementierung einer Baumstruktur für Verzeichnisse und Mengen.

Es gibt in der Baumstruktur zwei Arten von Objekten, Blätter und innere Knoten. Information ist nur in den inneren Knoten gespeichert, und zwar werden dort Schlüssel–Wert–Paare gespeichert.

Um eine binäre Suche durchführen zu können, muß auf den Schlüsseln eine Ordnungsfunktion definiert sein. Diese wird mittels einer Schnittstelle festgelegt.

Mit Prädikaten kann man Eigenschaften von einem Baum erfragen:

- **isEmpty** soll für den leeren Baum gelten
- **isIn** soll testen, ob ein Schlüssel in einem Baum gespeichert ist
- **isEqual** soll zwei Bäume auf Gleichheit testen. Zwei Bäume sind gleich, wenn sie die gleiche Struktur besitzen und an jedem Knoten die gleichen Werte gespeichert sind.

Die **left**– und **right**–Methoden sollen die linken und rechten Teilbäume eines Baums liefern.

lookup soll für einen Schlüssel das zugehörige Attribut berechnen, ist der Schlüssel nicht im Baum enthalten, soll **null** zurückgegeben werden.

noOfNodes berechnet die Anzahl der gespeicherten Paare, insert trägt in einen Baum ein neues Paar ein und liefert einen neuen Baum, der alte Baum wird also nicht verändert.

Teile der Implementierung sind vorgegeben, füllen sie die fehlenden Methodenrümpfe so, dass die oben geforderte Funktionsweise sichergestellt ist.

Die Schnittstelle für die Vergleichsfunktion, als Resultat soll ausschließlich -1, 0 und +1 geliefert werden. Ansonsten soll diese Funktion analog zur **strcmp**–Routine aus C arbeiten.

```
interface Comparable {
    int compare(Comparable c2);
}
```

Die **Tree**–Klasse und ihre Hilfsklassen:

```
public
    abstract
    class Tree {

        //——————————
        // Prädikate

        public abstract
            boolean isEmpty();

        public
            boolean isIn(Comparable k) {
            return
                lookup(k) != null;
        }

        public abstract
            boolean isEqual(Tree v2);

        //——————————
        // Selektoren

        public
            Tree left() {
            throw
                new RuntimeException("left not supported");
        }

        public
            Tree right() {
            throw
                new RuntimeException("right not supported");
        }
```

```
public
    Object lookup(Comparable c) {
    return
        null;
}

//—————————–

public abstract
    Tree insert(Comparable k, Object a);

public abstract
    int noOfNodes();

//—————————–

private static final
    Tree empty = new EmptyTree();

public static
    Tree makeEmpty() {
    return
        empty;
}

public static
    Tree makeOne(Comparable k, Object a) {
    return
        new Node(empty,empty,k,a);
}
```

```
//—————————
// die innere Klasse EmptyTree

private static final
    class EmptyTree extends Tree {

        public
            boolean isEmpty() {

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

                ..........................
        }
        public
            boolean isEqual(Tree v2) {

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

                ..........................
        }

        public
            Tree insert(Comparable k, Object a) {

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

                ..........................
        }

        public
            int noOfNodes() {

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

                ..........................
        }

// end class EmptyTree
//—————————
```

```
//—————————
// die innere Klasse Node

private static final
    class Node extends Tree {

        final
            Tree l, r;
        final
            Comparable k;
        final
            Object a;

        Node(Tree l, Tree r, Comparable k, Object a) {
            this.l = l;
            this.r = r;
            this.k = k;
            this.a = a;
        }

        public
            boolean isEmpty() {

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

             ..........................
        }

        public
            Tree left() {
            return
                 l;
        }

        public
            Tree right() {
            return
                 r;
        }
```

```
public
    boolean isEqual(Tree v2) {

        ..........

        ..........

        ..........

        ..........

        ..........

        ..........

        ..........

        ..........
    }
}

public
    Object lookup(Comparable k) {

        ..........

        ..........

        ..........

        ..........

        ..........

        ..........

        ..........

        ..........

        ..........

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

```
        public
            Tree insert(Comparable k, Object a) {

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

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

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

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

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

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

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

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

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

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

                ..........................
            }

            public
                int noOfNodes() {

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

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

                ..........................
            }
        }

    // end Node
    //---------

}
```

Gegeben sei die folgende stark vereinfachte Schnittstelle für eine Klasse für Mengen:

```
interface Set {

    // testet auf leere Menge
    public
        boolean isEmpty();

    // testet auf Enthaltensein
    public
        boolean isIn(Comparable c);

    // fügt ein Element in die Menge ein
    public
        void add(Comparable e);

    // berechnet die Länge einer Liste
    public
        int card();
}
```

Für diese Schnittstellen ist eine Implementierung zu entwickeln, die die `Tree`–Klasse verwendet. Der Rahmen für diese Klasse ist wieder vorgegeben. Entwickeln Sie die fehlenden Teile.

```
public
    class SetAsTree implements Set {
        private Tree t;

        public SetAsTree() {
            t = Tree.makeEmpty();
        }

        public
            boolean isEmpty() {

                ..........................
        }

        public
            boolean isIn(Comparable c) {

                ..........................
        }

        public
            void add(Comparable e) {

                ..........................
        }

        public
            int card() {

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

Aufgabe 2:

Gegeben sei das folgende Programm:

```
#include <stdio.h>

char * tab [] = { "Falle" , "Maus" , "Kaefig" , "Gericht" };

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

char * * * ppp = ptab;

int main ( int argc , char * argv [] )
{
  printf( "%s\n" , * (* (ppp + 2) - 1) + 1 );
  printf( "%s\n" , ppp [2] [0] + 3 );
  printf( "%s\n" , * (* ++ppp + 1) + 2 );
  printf( "%s\n" , * * ppp + 4 );

  return 0;
}
```

Welche Ausgabezeilen liefert dieses Programm:

1) .....

2) .....

3) .....

4) .....

Wieviel 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 3:

Gegeben sei das folgende Programm:

```
#include <ctype.h>
#include <stdio.h>

typedef char (*Translate)(char);

char ident(char c) {
  return c;
}

char toUpper(char c) {
  return islower(c) ? c - ('a' - 'A') : c;
}

char toLower(char c) {
  return isupper(c) ? c + ('a' - 'A') : c;
}

char toAlnumSpace(char c) {
  return isalnum(c) ? c : ' ';
}

char whiteSpaceToSpace(char c) {
  return isspace(c) ? ' ' : c;
}

char cntrlToSpace(char c) {
  return iscntrl(c) ? ' ' : c;
}

char spaceToMinus(char c) {
  return c == ' ' ? '-' : c;
}

char * translateString(char* d, char * s, Translate tf) {
  char * d1 = d;

  while (*s) {
    *d1++ = tf(*s++);
  }

  *d1 = 0;
  return d;
}
```

```
char * translateString2(char* d, char * s, Translate * ptf) {
  char * d1 = d;
  char ch;

  while ((ch = *s++)) {

    Translate * ptf1 = ptf;
    Translate tf;

    while ((tf = *ptf1++)) {
      ch = tf(ch);
    }
    *d1++ = ch;
  }
  *d1 = 0;
  return d;
}

char * tab[] = {
  "-abc-123-",
  "+\n+\n+\n+\n",
  "123 456",
  "a+-+-+z"
};

Translate t1[] = {whiteSpaceToSpace, spaceToMinus, (Translate)0};
Translate t2[] = {whiteSpaceToSpace, toUpper, (Translate)0};

#define OUT printf("%s\n", res)

int main(int argc, char* argv[]) {
  char res[256];

  translateString(res, tab[0], ident); OUT;

  translateString(res, tab[3], toUpper); OUT;

  translateString(res, tab[1], whiteSpaceToSpace);
  translateString(res, res, spaceToMinus); OUT;

  translateString2(res, tab[0], t2); OUT;

  translateString2(res, tab[1], t1); OUT;

  return 0;
}
```

Welche Ausgabezeilen liefert dieses Programm:

1) .....

2) .....

3) .....

4) .....

5) .....
