Fachhochschule Wedel
Prof. Dr. Uwe Schmidt

9. August 2002

Aufgaben zur Klausur **C** und **Objektorientierte Programmierung** im SS 2002 (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 19 Seiten

---

Aufgabe 1:

Welchen syntaktischen Einheiten in der Sprache C wird ein Typ zugeordnet?

1) .....

2) .....

3) .....

4) .....

5) .....

6) .....

Welchen syntaktischen Einheiten in der Sprache Java wird ein Typ zugeordnet?

1) .....

2) .....

3) .....

4) .....

5) .....

6) .....

Wann wird der Typ eines Ausdrucks in der Sprache C berechnet?

nur zur Laufzeit ☐

nur zur Übersetzungszeit ☐

weder zur Laufzeit noch zur Übersetzungszeit ☐

sowohl zur Laufzeit als auch zur Übersetzungszeit ☐

Wann wird der Typ eines Ausdrucks in der Sprache Java berechnet?

nur zur Laufzeit ☐

nur zur Übersetzungszeit ☐

weder zur Laufzeit noch zur Übersetzungszeit ☐

sowohl zur Laufzeit als auch zur Übersetzungszeit ☐

Welche Typkonstruktoren (welche Arten neue Typen zu erzeugt) gibt es in der Sprache C?

1) .....

2) .....

3) .....

4) .....

5) .....

6) .....

Welche unterschiedlichen Arten von Typen gibt es in der Sprache Java?

1) .....

2) .....

3) .....

4) .....

5) .....

6) .....

In welchen Ausdrücken in Java wird der Typ eines Objekts berechnet?

1) .....

2) .....

3) .....

4) .....

5) .....

6) .....

Wann werden in der Sprache C cast–Ausdrücke ausgewertet?

nur zur Laufzeit ☐

nur zur Übersetzungszeit ☐

weder zur Laufzeit noch zur Übersetzungszeit ☐

sowohl zur Laufzeit als auch zur Übersetzungszeit ☐

Wann werden in der Sprache Java cast–Ausdrücke ausgewertet?

nur zur Laufzeit ☐

nur zur Übersetzungszeit ☐

weder zur Laufzeit noch zur Übersetzungszeit ☐

sowohl zur Laufzeit als auch zur Übersetzungszeit ☐

In Java ist es möglich, dass in einer Klasse mehrere Methoden mit gleichem Namen existieren. Wann wird in **diesen** Fällen für einen Methodenaufruf entschieden, welche der Methoden aufgerufen wird?

nur zur Laufzeit ☐

nur zur Übersetzungszeit ☐

weder zur Laufzeit noch zur Übersetzungszeit ☐

sowohl zur Laufzeit als auch zur Übersetzungszeit ☐

Aufgabe 2:

Gegeben sei das folgende Programm:

```
#include <stdio.h>

char * tab [] = { "Dreyfuss" , "Kimball" , "dwelt" , "Kodachrome", "McAllister" };

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) + 4 );
  printf( "%s\n" , ppp [3] [0] + 3 );
  printf( "%s\n" , * (* (ppp + 2)) + 1 );
  printf( "%s\n" , * (* ++ppp) + 8 );
  printf( "%s\n" , * (* --ppp) + 5 );

  return 0;
}
```

Welche Ausgabezeilen liefert dieses Programm:

1) .....

2) .....

3) .....

4) .....

5) .....

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 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));
}

static unsigned int linecnt = 0;

#define PRINT(s) { printf("%2u) ", ++linecnt); printSet(s); printf("\n"); }

#define anInterval(n,m) (theFirst(m+1) ^ theFirst(n))
#define theFirst(n) (oneElement(n) - 1)
#define oneElement(i) ( (Set)(1 << (i)) )

int main(void) {
    Set s1;

    PRINT( oneElement(1) );
    PRINT( oneElement(SetMax-1) );
    PRINT( oneElement(SetMax) );

    PRINT( (Set)3 );
    PRINT( (Set)33 );

    PRINT( anInterval(1,SetMax-3) );
    PRINT( anInterval(2,2) );
    PRINT( anInterval(6,5) );

    PRINT( 29 & 28 );
    PRINT( 29 && 28 );

    s1 = ~anInterval(1,4) | anInterval(3,6); PRINT(s1);
    s1 = anInterval(1,6) ^ ((128 - 1) * 4); PRINT(s1);

    s1 = 32 + 24;
    s1 = 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 und numeriert die Zeilen durch.

Welche 13 Ausgabezeilen erzeugt dieses Programm?

1) .....

2) .....

3) .....

4) .....

5) .....

6) .....

7) .....

8) .....

9) .....

10) .....

11) .....

12) .....

13) .....

Aufgabe 4:

Die folgenden Klassen dienen zur Implementierung einer allgemeinen dynamischen Datenstruktur für Listen und Bäume. Die Bezeichnungen sind an die in LISP üblichen Bezeichnungen angelehnt. In dieser einfachsten universellen Datenstruktur gibt es drei Ausprägungen:

Atome sind die elemetaren Objekte, diese werden durch einen Namen identifiziert.

Ein spezielles elementares (atomares) Objekt wird mit nil bezeichnet und wird z.B. für die Darstellung einer leeren Liste verwendet.

Zusammengesetzte Objekte werden mit Hilfe von Paaren aufgebaut, wobei die beiden Teile wieder beliebig komplexe Werte sein dürfen.

Listen werden üblicherweise als eine Folge von Paaren dargestellt, wobei die Elemente der Liste über den cdr-Verweis verkettet werden und das Ende der Liste durch eine Referenz auf das nil-Objekt gekennzeichnet wird.

Mit Prädikaten kann man Eigenschaften von einem Wert erfragen:

- isAtom soll gelten für nicht zusammengesetzte Objekte
- isNil nur für den speziellen nil–Wert
- isPair nur für zusammengesetzte Werte
- isList soll gelten für die leere Liste, repäsentiert durch nil, und für Paare, deren zweiter Teil (cdr) eine Liste ist
- isEqual soll zwei beliebige Werte auf Gleichheit überprüfen. Zu implementieren ist dieser Test mit möglichst wenigen Vergleichsoperationen.

Die append-Methode soll eine neue Liste aufbauen, bei der der Wert am Ende der Liste steht, also über die Referenz in car aus dem letzten Paar zugreifbar ist.

Die length Methode berechnet die Anzahl der Paare, die über die cdr-Referenzen verkettet sind.

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

Die toString–Methoden sind als reine Testmethoden zu behandeln, keine der anderen Methoden sollte diese in ihrer Implementierung nutzen.

```
public
    abstract
    class Value {

        //---------
        // Prädikate

        public
            boolean isAtom() {
            return
                false;
        }
        public
            boolean isNil() {
            return
                false;
        }
        public
            boolean isPair() {
            return
                false;
        }
        public
            boolean isList() {
            return
                false;
        }
        public
            boolean isEqual(Value v2) {
            return
                false;
        }

        //---------
        // Selektoren

        public
            Value car() {
            throw
                new RuntimeException("car not supported");
        }

        public
            Value cdr() {
            throw
                new RuntimeException("cdr not supported");
        }
```

```
//—————————

public
    Value append(Value v2) {
    return
        new Pair(v2,this);
}


public
    int length() {
    return
        0;
}

//—————————

public static final
    Value nil = new Nil();

public static
    Value atom(String name) {
    return
        new Atom(name);
}

public static
    Value pair(Value car, Value cdr) {
    return
        new Pair(car,cdr);
}
```

```
//innere Klasse Nil

private static final
    class Nil extends Value {

        public
            boolean isAtom() {

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

                ..........................
        }
        public
            boolean isNil() {

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

                ..........................
        }
        public
            boolean isList() {

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

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

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

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

        public
            String toString() {
            return
                "nil";
        }
    }

// Ende Klasse Nil
```

```
// innere Klasse Atom

private static final
    class Atom extends Value {

        final
            String name;

        Atom(String name) {
            this.name = name;
        }

        public
            boolean isAtom() {

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

        public
            boolean isEqual(Value v2) {

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

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

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

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

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

        public
            String toString() {
            return
                 name;
        }
    }

// Ende Klasse Atom
```

```
// innere Klasse Pair

private static final
    class Pair extends Value {

        final
            Value car, cdr;

        Pair(Value car, Value cdr) {
            this.car = car;
            this.cdr = cdr;
        }

        public
            boolean isPair() {

             ..........................
        }
        public
            boolean isList() {

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

        public
            Value car() {
            return
                car;
        }

        public
            Value cdr() {
            return
                cdr;

        public
            String toString() {
            return
                "( " + car.toString() + "  .   " + cdr.toString() + " )";
        }
```

```
        public
            boolean isEqual(Value v2) {

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

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

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

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

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

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

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

            public
                Value append(Value v2) {

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

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

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

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

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

            public
                int length() {

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

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

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

                ..........................
            }
        }
    // Ende Klasse Pair
}
```

Fragen:

1. Ist das private Attribut für die inneren Klassen Nil, Atom und Pair wichtig?

   ja ☐ nein ☐

   Begründung:

   .....

2. Ist das static Attribut für die inneren Klassen Nil, Atom und Pair wichtig?

   ja ☐ nein ☐

   Begründung:

   .....

3. Würden diese Klassen auch noch korrekt arbeiten, wenn man das static Attribut weglässt?

   ja ☐ nein ☐

   Begründung:

   .....

4. Hätte das fehlende static Attribut Einfluss auf die Größe der Objekte?

   ja ☐ nein ☐

   Begründung:

   .....

5. Ist die Funktion atom in der Klasse Value notwendig, oder ist es vernünftiger, den sehr kurzen Funktionsrumpf in die Anwendungen direkt einzukopieren.
   Notwendig?

   ja ☐ nein ☐

   Begründung:

   .....

6. Was müßte man ändern, damit man diese atom Funktion einsparen kann.

   .....

7. Wäre eine entsprechende Zugriffsfunktion für das Spezialobjekt *nil* in folgender Form public static $Value\ nil()\{$return $nil;\}$ aus software-technischen Gründen sinnvoll?

   ja ☐ nein ☐

   Begründung:

   .....

8. Ist das final Attribut für die static Variable nil wichtig?

   ja ☐ nein ☐

   Begründung:

   .....

9. Ist das final Attribut für die inneren Klassen Nil, Atom und Pair wichtig?

   ja ☐ nein ☐

   Begründung:

   .....

10. Ist es aus softwaretechnischen Gründen sinnvoll, die Klassen Nil, Atom und Pair als innere Klassen zu realisieren?

    ja ☐ nein ☐

    Begründung:

    .....

    .....

11. Wieviele Objekte aus der Klasse Nil werden in einem Programm, das diese Klassen verwendet, erzeugt?

    .....

12. Mit welcher Zeitkomplexität arbeitet die append-Methode?

    .....

13. Mit welcher Speicherplatzkomplexität arbeitet die append Methode?

    .....

14. Mit welcher Speicherplatzkomplexität arbeitet die statische pair Methode aus Value?

    .....

15. Mit welcher Zeitkomplexität arbeitet die isEqual-Methode?

    .....

Aufgabe 5:

Gegeben sei das folgende Java–Programmstück. Es soll überprüft werden ob die Zuweisungen und Konversionen legal sind. Hierbei sind drei Fälle zu unterscheiden:

1. Die Korrektheit der Konversion kann statisch zur Übersetzungszeit überprüft werden und ist erlaubt.
   Dies ist zu kennzeichnen durch Ankreuzen von ☐ ct.
2. die Korrektheit der Konversion wird zur Laufzeit überprüft.
   Dies ist zu kennzeichnen durch Ankreuzen von ☐ rt.
3. es kann zur Übersetzungszeit festgestellt werden, daß die Konversion fehlerhaft ist.
   Dies ist zu kennzeichnen durch Ankreuzen von ☐ err.

```
interface X { }

interface Y extends X { }

class Base { }

class D1 extends Base { }

class E1 extends D1 implements Y { }

class D2 extends Base implements X { }
```

```
class Conversion {
    Object o;
    Base b;
    D1 d1;
    E1 e1;
    D2 d2;
    X x;
    Y y;
    Object [] oa;
    Base [] ba;
    D1 [] da;

    void f() {
        d2 = (D2)b;                    ct  rt  err
        d2 = (D2)d1;                   ct  rt  err
        d2 = (D2)x;                    ct  rt  err
        d2 = (D2)o;                    ct  rt  err

        d1 = (D1)ba;                   ct  rt  err
        d1 = (D1)(ba[0]);              ct  rt  err
        d1 = (D1)e1;                   ct  rt  err

        x = (X)b;                      ct  rt  err
        x = (X)da;                     ct  rt  err
        x = (X)(da[0]);                ct  rt  err
        x = (X)e1;                     ct  rt  err
        x = (X)y;                      ct  rt  err

        oa = (Object [])b;             ct  rt  err
        oa = (Object [])ba;            ct  rt  err
        oa = (Object [])o;             ct  rt  err
    }
}
```