Fachhochschule Wedel
Prof. Dr. Uwe Schmidt

2. Februar 1998

Aufgaben zur Klausur **C** und **Objektorientierte Programmierung** im WS 98/99 (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 13 Seiten

---

Aufgabe 1:

Gegeben sei das folgende Programm

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

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

    char *x[10];
    char **p = x;
    char **q = p;

    if ( strcmp(*p++, *q++) )
        printf("gleich");
    else
        printf("nicht gleich");

    return 0;
}
```

Welche Ausgabe erzeugt dieses Programm?

. . . . . .

Warum?

. . . . . .

. . . . . .

Aufgabe 2:

Gegeben seien die folgenden Variablen:

int x;
unsigned int u;
long int s;
float f;
char *p1;
int *p2;
void *p3;

Bestimmen Sie für die folgenden Ausdrücke den Typ gemäß ANSI-C. Vorsicht: Es kommen fehlerhafte Ausdrücke von. Kennzeichnen Sie diese entsprechend

++f .....

p2[x] .....

p1 ? x : f .....

! p3 .....

p1 == p3 .....

p1 = p3 .....

~p2 .....

p1 && p1 .....

x += f .....

~s .....

! s .....

p3 ? x : f .....

s || x .....

p1 + 0x23 .....

s | x .....

Aufgabe 3:

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 I { }

interface J extends I { }

class X { }

class Y extends X { }

class Z extends Y implements J { }

class U extends X implements I { }
```

```
class Conversion {
    Object o; Object [] oa;

    X x; Y y; Z z; U u; I i; J j;

    X [] xa; Y [] ya;

    void f() {

        oa = (Object [])xa;        ct  rt  err
        oa = (Object [])o;         ct  rt  err
        oa = (Object [])x;         ct  rt  err

        u = (U)o;                  ct  rt  err
        u = (U)x;                  ct  rt  err
        u = (U)y;                  ct  rt  err
        u = (U)i;                  ct  rt  err

        y = (Y)z;                  ct  rt  err
        y = (Y)xa;                 ct  rt  err
        y = (Y)(xa[0]);            ct  rt  err

        i = (I)j;                  ct  rt  err
        i = (I)x;                  ct  rt  err
        i = (I)z;                  ct  rt  err
        i = (I)ya;                 ct  rt  err
        i = (I)(ya[0]);            ct  rt  err
    }
}
```

Aufgabe 4:

Gegeben sei das folgende Java–Programm:

```
class X {
    int i;

    X ref;

    X() { i = 0; ref = this; }

    void out() { System.out.println("i = " + i); }

    void f() { i = 1; }

    void g() { ++i; }

    void h() { f(); ref.g(); }
}

class Y extends X {
    void f() { i = 2; }
}

class Z extends Y {
    void g() { i += 3; }
}

class U extends X {
    void h() { super.h(); ref.h(); }
}

public class Virtual {
    public static void main(String [] argv) {
        X a = new X();
        X b = new Y();
        X c = new Z();
        X d = new U();

        a.h(); a.out();
        b.h(); b.out();
        c.h(); c.out();
        d.h(); d.out();
    }
}
```

Welche Ausgabezeilen erzeugt dieses Programm?

1) .....

2) .....

3) .....

4) .....

Aufgabe 5:

Gegeben seien die folgende Schnittstelle

```
public interface List {
    public abstract Object head();

    public abstract List tail();

    public abstract int len();

    public abstract boolean isEmpty();

    public abstract List prepend(Object e);

    public abstract List append(Object e);
}
```

und die Ausnahme–Klasse

```
public class ListException extends RuntimeException {
    ListException(String err) {
        super(err);
    }
}
```

Diese Schnittstelle für einen einfachen ADT für Listen soll mit der folgenden Klasse *LinkedList* implementiert werden. *head*, *tail*, *len* und *isEmpty* sind die üblichen Zugriffsfunktionen, *prepend* soll ein Element vorne an die Liste anhängen, *append* soll eine neue(!!!) Liste aufbauen, die entsteht, wenn man ein Element hinten an die Liste anhängt. Wenn Operationen nicht ausführbar sind, soll eine Ausnahme ausgelöst werden.

Entwickeln Sie die Methodenrümpfe für die einzelnen Methoden für die *LinkedList* Klasse und die beiden lokalen(!!!) Klassen *Empty* und *Node*. Bei dieser Aufgabe ist es notwendig, zuerst die Klassenstruktur vollständig zu verstehen, bevor ein Aufgabenteil gelöst wird. Hinweis: es gibt zwei Arten von Listen, die leere Liste und Listen mit mindestens einem Element.

Wählen Sie sinnvolle Zugriffsattribute für die Klassen und die einzelnen Datenfelder und Methoden aus den Attributen **public**, **protected**, **private** oder als Default *friendly* aus. Versuchen Sie dabei, die Sichtbarkeit so weit wie möglich einzuschränken.

```
.............
    abstract class LinkedList implements List {


        .............
            static final List empty = new Empty();


        .............
            static List mkEmptyList() {

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

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


        .............
            static List mkOneElementList(Object e) {

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

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


        .............
            static final class Empty extends LinkedList {

                .............
                    Object head() {

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

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

```
// LinkedList.Empty (cont.)

        ..............
            List tail() {

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

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


        ..............
            int len() {

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

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


        ..............
            boolean isEmpty() {

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

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


        ..............
            List prepend(Object e) {

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

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


        ..............
            List append(Object e) {

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

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

```
// LinkedList.Node

        ..............
            static final class Node extends LinkedList {


                ..............
                    Object info;


                ..............
                    List next;


                ..............
                    Node(Object e, List l) {
                    info = e;
                    next = l;
                }


                ..............
                    Object head() {

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

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


                ..............
                    List tail() {

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

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

```
// LinkedList.Node (cont.)

                .....
                    int len() {

                    .....

                    .....

                    .....
                }

                .....
                    boolean isEmpty() {

                    .....

                    .....
                }

                .....
                    List prepend(Object e) {

                    .....

                    .....
                }

                .....
                    List append(Object e) {

                    .....

                    .....

                    .....
                }
            }
    }
```
