Physikalisch–Technische Lehranstalt Wedel
Prof. Dr. Uwe Schmidt

13. Februar 1997

Aufgaben zur Klausur **Objektorientierte Programmierung** im WS 96/97 (IA42)

Zeit: 60 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.

Viel Erfolg !

Diese Klausur besteht einschließlich dieses Deckblattes aus 9 Seiten

---

Aufgabe 1:

Gegeben sei das folgende C++ Programmstück:

```
// header Dateien fuer Ausnahmebehandlung

#include <std/typeinfo.h>
#include <iostream.h>

// zwei Ausnahmen fuer Fehlersituationen

class HeapOverflow {};
class IndexError {};

// eine Klasse fuer dynamische Felder

class Field {

private:
  double * a;

protected:
  unsigned len;

private:
  unsigned check(unsigned i) const;

public:
  Field(unsigned l)
    : len(l) {
      a = new double[l];
      if ( a == 0 )
        throw HeapOverflow();
  }

  ~Field() {
    delete [] a;
  }

  double operator [] (unsigned i) const {
    return a[check(i)];
  }
  void putAt(unsigned i, double v) {
    a[check(i)] = v;
  }
};
```

```
//--------------------------

// eine Klasse für allgemein nuetzliche arithmetische Operationen auf Feldern


class Arithmetic {
public:
   virtual double sum() const = 0;
   virtual unsigned card() const = 0;
   virtual double arithmeticMean() const {
      return sum() / card();
   }
};

//--------------------------
```

Entwickeln Sie die fehlenden Routinen für die Klasse *Field*. Dabei sollen auftretende Fehlersituationen analog zu den vorgegebenen Programmteilen behandelt werden.

Entwickeln Sie eine Klasse *Field1*, die alle Eigenschaften der Klasse *Field* besitzt, außerdem aber noch die Funktionen aus der Klasse *Arithmetic* bereitstellt.

Aufgabe 2:

Gegeben seien die folgenden C++ Programmstücke:

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

class Ordnung {
public:
  virtual int operator == (const Ordnung & o2) const =0;
  virtual int operator != (const Ordnung & o2) const {
    return ! ( *this == o2 );
  }
  virtual int operator ≥ (const Ordnung & o2) const =0;
  virtual int operator ≤ (const Ordnung & o2) const {
    return ( o2 ≥ *this );
  }
  virtual int operator > (const Ordnung & o2) const {
    return ( *this ≥ o2 ) && ! ( *this == o2 );
  }
  virtual int operator < (const Ordnung & o2) const {
    return ( o2 > *this );
  }
};

class Str : public Ordnung {
private:
  char * p;

  void init(const char * s) {
    p = new char[strlen(s) + 1];
    strcpy(p,s);
  }
  void del() {
    delete [] p;
  }
public:
  Str() {
    init("");
  }
  Str(const char * s) {
    init(s);
  }
  Str(const Str & s) {
    init(s.p);
  }
  Str & operator = (const Str & s1) {
    if ( this != &s1 ) { del(); init(s1.p); }
```

```
    return *this;
  }
  virtual ~Str() {
    del();
  }
  virtual int operator == (const Ordnung & o2) const {
    return strcmp(p,((const Str&)o2).p) == 0;
  }
  virtual int operator ≥ (const Ordnung & o2) const {
    return strcmp(p,((const Str&)o2).p) ≥ 0;
  }
};
```

Entwickeln Sie hieraus eine konkrete Klasse *Person*: eine Person soll durch 2 strings für Name und Vorname und einem ganzzahligen Datenfeld für das Alter bestehen. Die Klasse soll ebenfalls die Ordnungsoperationen aus der Klasse Ordnung zur Verfügung stellen, wobei der Nachname für die Ordnung wichtiger als der Vorname ist. Bei Namensgleichheit sollen ältere Personen "größer als" jüngere sein. Es soll nicht mit Mehrfachvererbung gearbeitet werden. Achten Sie beim Entwurf des Konstruktors und Destruktors darauf, daß nicht überflüssige Kopien gemacht werden, daß aber die Objekte auch nicht von Destruktoren aus der Str-Klasse zerstört werden können.

der Klassenkopf und die Datenkomponenten

.....

.....

.....

.....

.....

.....

ein geeigneter Konstruktor, der aus den 3 Komponenten ein Objekt erzeugt, und ein Destruktor

.....

.....

.....

.....

.....

.....

die Ordnungsfunktionen