Programmieren in C++ Sommersemester 1997
Klausur mit Musterlösung, 3.7.1997

Vorname	Name	Matr.-Nr.	Email	Bonus	Klausur	Summe
				/7.5	/50	

1. Was ist ... (Bitte knappe, präzise Antworten) [10 Punkte]

a) Eine abstrakte Klasse?

Eine Klasse, die mindestens eine pur virtuelle Methode enthält.

b) Eine Schablonenfunktion?

Eine durch das `template`*-Konstrukt generisch typisierte Funktion.*

c) Eine virtuelle Basisklasse?

Eine Klasse, deren Datenelemente trotz mehrfacher Vererbung (indirekt auf verschiedenen Wegen) nur einmal geerbt werden.

d) Ein Klassenattribut?

Ein Attribut, das Eigenschaften einer Klasse beschreibt. Es exisitiert nur einmal für die Klasse. Definition in C++ mit "static".

e) Dynamische Bindung?

Die Entscheidung, welche Methode aufzurufen ist, fällt zur Laufzeit anhand des Typs des Objektes, auf das die Methode angewandt wird.

2. Welche Ausgabe erzeugt ... [10 Punkte]

```
#include <iostream.h>

class bruch
{
private:
    int nenn;
    int zaehl;
public:
    bruch(int z=0, int n=1): zaehl(z), nenn(n) {test('K');}
    bruch(const bruch& b): nenn(b.nenn), zaehl(b.zaehl) {test('C');}
    ~bruch() {test('D');}
    void test(char w) {cout <<  w << ": " << zaehl << '/' << nenn << endl;}
};

bruch gen7()
{
    bruch result(7,1);
    return result;
}

int main()
{
    bruch cb(1);
    bruch ba[3] = {bruch(2,3), cb};
    bruch *bp = new bruch[2];
    gen7().test('M');
}

{\bf Lösung}:   K: 1/1                    <erste zeile>
          K: 2/3                    <zweite zeile>
          C: 1/1
          K: 0/1
          K: 0/1                    <dritte zeile>
          K: 0/1
          K: 7/1                    <vierte zeile>
          C: 7/1
          D: 7/1
          M: 7/1
          D: 7/1
          D: 0/1                    <destruktion>
          D: 1/1
          D: 2/3
          D: 1/1
```

3. Ergänzen Sie ... [10 Punkte]

```
class vector
{   double *co;
    int dim;
public: vector(int d) : dim(d), co(new double[d]) {}
        double& operator [] (int d) {return co[d];}
};

int main()
{ vector v(10);
  for (int i = 1; i <= 10; i++)
    v[i-1] = i;
  double sum = v;
  cout << sum << endl;
}
```

An der Stelle `double sum = v;` meldet der Compiler:

```
Error: Cannot use vector to initialize double.
```

Ergänzen Sie die Klasse `vector`, so daß dieser Fehler nicht mehr auftritt und `sum` mit der Summe der Elemente im Vektor `v` (hier 55) initialisiert wird.

Lösung (Typumwandlungsoperator `vector` nach `double`):

innerhalb der Klassendefinition ergänzen:

```
operator double()
    {
        double sum = 0;
        for (int d=0; d < dim; d++)
            sum += co[d];
        return sum;
    }
```

4. Implementieren Sie ... [10 Punkte]

die Klasse `Termin`, die wie folgt entworfen worden ist:

- Es existiert eine Klasse `Uhrzeit`

```
class Uhrzeit
{ private: ...
  public: Uhrzeit();
          Uhrzeit (int std, int min);
          void print();
};
```

- Es existiert eine Klasse `Date`

```
class Date
{ private: ...
  public: Date();
          Date (int tag, int monat, int jahr);
          void print();
};
```

- Ein `Termin` *ist ein* `Date`.
- Ein `Termin` *hat eine* `Uhrzeit` und ein `Thema` (repräsentiert als `char*`).

Geben Sie eine Definition der Klasse `Termin` an mit einem Default-Konstruktor `Termin()`, einem dreistelligen Konstruktor `Termin(Date, Uhrzeit, char*)` und einer Methode `print()`.

Lösung:

```
class Termin : public Date
{
private:
    Uhrzeit zeit;
    char *thema;
public:
    Termin() : Date(), zeit(), thema("")
    {
    }
    Termin(Date d, Uhrzeit u, char *t) : Date(d), zeit(u), thema(t)
    {
    }
    void print()
    {
     cout << "Termin am: "; zeit.print(); Date::print();
     cout << "Thema: << thema";
    }
};
```

5. Gegeben ist ... [10 Punkte]

die folgende Template-Deklaration:

```
template <class T>
class arr
{ protected: T* _inhalt;
             int _elementanzahl;

  public:    arr(int anz=10) : _elementanzahl(anz), _inhalt(new T[anz]) {}
             arr(const arr<T>&);                   // Kopie
             ~arr();                               // Destruktion
             arr<T>& operator= (const arr<T>&);    // Wertzuweisung
             T& operator [] (int i) {return _inhalt[i];}
             int elementanzahl() const {return _elementanzahl;}
};
```

Definieren Sie die drei fehlenden Elemente der Schablonen-Klasse `arr` für Kopie, Destruktion und Wertzuweisung außerhalb der Klassendefinition. Denken Sie bei der Wertzuweisung an die notwendige Speicherfreigabe.

Lösung

```
// Kopie
template <class T>
arr<T>::arr(const arr<T>& ori): _elementanzahl(ori._elementanzahl)
{   _inhalt = new T[ori._elementanzahl];
    for (int i=0; i < _elementanzahl; i++)
        _inhalt[i] = ori._inhalt[i];
}
// Destruktion
template <class T>
arr<T>::~arr()
{ delete[] _inhalt;
}
// Wertzuweisung
template <class T>
arr<T>& arr<T>::operator=(const arr<T>& right)
{   if (this != &right)
    {   delete[] _inhalt;
        _inhalt = new T[right._elementanzahl];
        _elementanzahl=right._elementanzahl;
        for (int i=0; i < _elementanzahl; i++)
            _inhalt[i] = right._inhalt[i];
    }
    return *this;
}
```