

# Übung05: Informationstechnik (IT)

**Institutsleitung**  
Prof. Dr.-Ing. K. D. Müller-Glaser  
Prof. Dr.-Ing. J. Becker  
Prof. Dr. rer. nat. W. Stork

**Tobias Schwalb & Timo Sandmann & Stephan Werner**

Institut für Technik der Informationsverarbeitung (ITIV)



## Teil1: Besprechung der Übungsaufgaben 4.01-4.04

KIT – Universität des Landes Baden-Württemberg und  
nationales Forschungszentrum in der Helmholtz-Gemeinschaft

[www.kit.edu](http://www.kit.edu)

## Inhalt: Übung05 – Teil 1

- Aufg. 4.01: Verständnisfragen
- Aufg. 4.02: Dynamische Arrays
- Aufg. 4.03: OOP, Definition von Klassen
- Aufg. 4.04: OOP, Klassendefinition

## Aufg. 4.01: Verständnisfragen Lsg. (1)

- a) Der Operator `new` erwartet als Operand
  - i. den Namen des Objekts, das dynamisch erzeugt werden soll. Falsch
  - ii. den Typ des anzulegenden Objekts. Richtig
  - iii. die Größe des Objekts in Anzahl Bytes. Falsch
  
- b) Mit `new` reserviertem Speicher wird mit dem Operator `delete` wieder freigegeben.
  
- c) Wird für ein dynamisch reserviertes Objekt der `delete`-Operator nicht aufgerufen, so wird der dynamisch reservierte Speicherbereich
  - i. nicht wieder durch das Programm freigegeben. Richtig
  - ii. automatisch freigegeben, wenn er nicht mehr verwendet wird. Falsch

## Aufg. 4.01: Verständnisfragen Lsg. (2)

- d) Um den kontrollierten Zugriff auf die Datenelemente einer Klasse sicherzustellen, werden die Datenelemente normalerweise als `private` deklariert und durch `get` & `set-Methoden` gelesen oder geändert.
  
- e) Innerhalb einer Methode der Klasse können folgende Elemente dieser Klasse direkt mit ihrem Namen angesprochen werden:
  - i. alle Elemente Richtig
  - ii. nur die Datenelemente Falsch
  - iii. nur die `public`-Elemente Falsch
  
- f) Um beim Aufruf einer Methode den für den Hin- und Rücksprung notwendigen Overhead zu vermeiden, kann eine Methode als `inline` definiert werden.

## Aufg. 4.01: Verständnisfragen Lsg. (3)

- g) Wenn zwei verschiedene Objekte derselben Klasse angelegt werden, wird der Maschinencode für jede Methode auch zweimal erzeugt.  
Falsch
- h) Bei einem Funktionsaufruf kann der durch das Erzeugen und Zerstören von Objekten bedingte Overhead vermieden werden, wenn Argumente per Referenz oder Zeiger übergeben werden.

## Aufg. 4.02: dynamische Arrays

- Schreiben Sie eine Funktion, welche elementweise die Summe zweier gleich langer Arrays in ein dynamisch reserviertes Array schreibt und einen Zeiger auf das neue Array zurückgibt. Der Typ der Arrayelemente ist **long**.
- Der Funktion werden beide Arrays und ihre gemeinsame Länge übergeben. Testen Sie die Funktion, indem Sie die Arrays mit Werten Ihrer Wahl initialisieren und die Summe der Arrays anzeigen. Geben Sie danach den Speicherplatz explizit frei.
- Beispielausgabe:

```
C:\Dokumente und Einstellungen\Admin\Eigene Dateien\CppV
Array 1: 324  4  45  66 345  43  9  34 111  88
Array 2:  42  5  23 244  53 355  35  33  1  35
Dy. Arr: 366  9  68 310 398 398  44  67 112 123
Arrayadresse: 00365B08
```

## Aufg. 4.02: dynamische Arrays Lsg. (1)

```

#include <iostream>
#include <iomanip>
using namespace std;

long* element_sum( long arr_1[], long arr_2[], int laenge );

int main() {
    long wert_1[10] = {324, 4, 45, 66, 345, 43, 9, 34, 111, 88};
    long wert_2[10] = {42, 5, 23, 244, 53, 355, 35, 33, 1, 35};
    long* summ = NULL;

    cout << "Array 1: ";
    for( int i = 0; i < 10; i++ ) {
        cout << setw( 4 ) << wert_1[i];
    }

    cout << endl << "Array 2: ";
    for( int i = 0; i < 10; i++ ) {
        cout << setw( 4 ) << wert_2[i];
    }
}

```

Zum Speichern der Adresse des dynamischen Arrays

Array1 ausgeben

Array2 ausgeben

setw(): Legt die Breite eines Ausgabefelds fest  
 → Einbinden der Bibliothek <iomanip> ist erforderlich

## Aufg. 4.02: dynamische Arrays Lsg. (2)

```

summ = element_sum( wert_1, wert_2, 10 );

cout << endl << endl << "dy. Arr: ";
for( int i = 0; i < 10; i++ ) {
    cout << setw( 4 ) << summ[i];
}

cout << endl << "Arrayadresse: " << summ << endl;

delete[] summ;
summ = NULL;
return 0;
}

long* element_sum( long arr_1[], long arr_2[], int laenge ) {
    long* p = new long[laenge];

    for( int i = 0; i < laenge; i++ ) {
        p[i] = arr_1[i] + arr_2[i];
    }

    return p;
}

```

Funktion aufrufen, Arrays und Länge werden übergeben

dynamisches Array ausgeben

Speicher des dynamisch erzeugten Arrays wieder freigeben

dynamisches Array anlegen

Summe bilden und in Array ablegen

Adresse des dynamisch erzeugten Arrays zurückgeben

```

C:\WINDOWS\system32\cmd.exe
Array 1: 324  4  45  66 345  43  9  34 111  88
Array 2:  42  5  23 244  53 355  35  33  1  35
dy. Arr: 366  9  68 310 398 398  44  67 112 123
Arrayadresse: 00365B80

```

## Aufg. 4.03: OOP, Definition von Klassen

- Definieren Sie eine Klasse PiggyBank zur Verwaltung der Münzen in einem Sparschwein.
- Die Klasse besitzt dabei private und öffentliche Attribute und Methoden
- Testen Sie die Klasse PiggyBank mit einem Anwendungsprogramm in einer separaten Quelldatei

```

C:\WINDOWS\system32\cmd.exe
Das Sparschwein ist leer!
Ersparnisse machen gluecklich!
Werfen Sie einige 1-Cent Stuecke ein: 123
Werfen Sie einige 10-Cent Stuecke ein: 220
Werfen Sie einige 50-Cent Stuecke ein: 77
Werfen Sie einige Euros ein: 90
Vielen Dank!

Das Sparschwein ist voll!
Das Sparschwein wird geschlachtet!
Im Sparschwein waren:
123 1-Cent Stuecke
220 10-Cent Stuecke
77 50-Cent Stuecke
00 1-Euro Stuecke
Das sind 141 Euros und 73 cents.

Gratuliere!
Drücken Sie eine beliebige Taste . . .
    
```

## Aufg. 4.03: OOP, Definition von Klassen Lsg. (1)

```

#ifndef _PIGGYBANK_
#define _PIGGYBANK_

class PiggyBank {
private:
    unsigned int nCent1, nCent10, nCent50, nEuro1;
    unsigned int max;
    bool fBroken;
public:
    PiggyBank( unsigned int m ) {
        max = m;
        nCent1 = nCent10 = nCent50 = nEuro1 = 0;
        fBroken = false;
    }
    bool isEmpty();
    bool isFull();
    bool isBroken();
    unsigned int add1Cents( unsigned int n1C );
    unsigned int add10Cents( unsigned int n10C );
    unsigned int add50Cents( unsigned int n50C );
    unsigned int add1Euros( unsigned int n1E );
    unsigned long breakInto( unsigned int& c1, unsigned int& c10,
                            unsigned int& c50, unsigned int& e1 );
    ~PiggyBank() {}
};
    
```

piggyBank.h

Präprozessor-Direktiven: Header-Datei wird nur einmal eingebunden

Maximale Anzahl an Münzen

für aufgebrochenes Sparschwein

Implizit inline Konstruktor

Methoden zum Überprüfen bestimmter Bedingungen anhand der privaten Attribute

Implizit inline dummy-Destruktor

## Aufg. 4.03: OOP, Definition von Klassen Lsg. (2)

```
//...
inline bool PiggyBank::isEmpty() {
    unsigned int cur = nCent1 + nCent10 + nCent50 + nEuro1;
    return cur == 0;
}

inline bool PiggyBank::isFull() {
    unsigned int cur = nCent1 + nCent10 + nCent50 + nEuro1;
    return cur >= max;
}

inline bool PiggyBank::isBroken() {
    return fBroken;
}

#endif
```

Definitionen der explizit inline Methoden der Klasse

inline Schlüsselwort nicht vergessen!

Methode durch :: Operator definieren

Ende des Präprozessor-Blocks  
Ende der Header-Datei

## Aufg. 4.03: OOP, Definition von Klassen Lsg. (3)

```
#include "piggyBank.h"

unsigned int PiggyBank::add1Cents( unsigned int n1C ) {
    unsigned int cur = nCent1 + nCent10 + nCent50 + nEuro1;
    if( n1C <= max - cur ) {
        nCent1 += n1C;
        return 0;
    } else {
        nCent1 += ( max - cur );
        return (n1C - ( max - cur ));
    }
}

unsigned int PiggyBank::add10Cents( unsigned int n10C ) {
    unsigned int cur = nCent1 + nCent10 + nCent50 + nEuro1;
    if( n10C <= max - cur ) {
        nCent10 += n10C;
        return 0;
    } else {
        nCent10 += ( max - cur );
        return n10C - ( max - cur );
    }
}
```

Enthält die Definition der Klasse

Falls alle Münzen ins Sparschwein passen

Sparschwein nimmt so viele Münzen wie möglich

Anzahl restlicher Münzen

piggyBank.cpp

## Aufg. 4.03: OOP, Definition von Klassen Lsg. (4)

```
unsigned int PiggyBank::add50Cents( unsigned int n50C ) {
    // analog zu den oberen Methoden
}

unsigned int PiggyBank::add1Euro( unsigned int n1E ) {
    // analog zu den oberen Methoden
}

unsigned long PiggyBank::breakInto( unsigned int& c1,
                                   unsigned int& c10,
                                   unsigned int& c50,
                                   unsigned int& e1 ) {

    fBroken = true;
    c1 = nCent1; c10 = nCent10; c50 = nCent50; e1 = nEuro1;
    nCent1 = nCent10 = nCent50 = nEuro1 = 0;
    unsigned long sum = c1 + c10 * 10 + c50 * 50 + e1 * 100;
    return sum;
}
```

Referenz Parameter

Münzenanzahl  
zurückgesetzt

Rückgabewert ist der Geldbetrag in Cents

## Aufg. 4.03: OOP, Definition von Klassen Lsg. (5)

```
// Testprogramm
#include "piggyBank.h"
#include <iostream>
using namespace std;

void showCoins( unsigned int c1, unsigned int c10,
               unsigned int c50, unsigned int e1 ) {
    if( c1 > 0 ) cout << c1 << " 1-Cent Stuecke" << endl;
    if( c10 > 0 ) cout << c10 << " 10-Cent Stuecke" << endl;
    if( c50 > 0 ) cout << c50 << " 50-Cent Stuecke" << endl;
    if( e1 > 0 ) cout << e1 << " 1-Euro Stuecke" << endl;
}

int main() {
    unsigned int c1 = 0, c10 = 0, c50 = 0, e1 = 0;
    PiggyBank myPiggy( 500 );

    if( myPiggy.isEmpty() ) {
        cout << "Das Sparschwein ist leer!" << endl;
    }
    //...
}
```

Enthält die Definition der Klasse

Funktion zur Ausgabe der  
gesammelten Münzen

Erzeugen eines Objekts der  
Klasse **PiggyBank** durch Aufruf  
des Konstruktors

Aufruf einer Methode der  
Instanz der Klasse **PiggyBank**

main.cpp

## Aufg. 4.03: OOP, Definition von Klassen Lsg. (6)

Solange das Sparschwein  
nicht voll ist

```
while( !myPiggy.isFull() ) {
    cout << "\nErsparnisse machen gluecklich!" << endl;
    cout << "Werfen Sie einige 1-Cent Stuecke ein: ";
    if( !( cin >> c1 ) || myPiggy.add1Cents( c1 ) != 0 ) break;

    cout << "Werfen Sie einige 10-Cent Stuecke ein: ";
    if( !( cin >> c10 ) || myPiggy.add10Cents( c10 ) != 0 ) break;

    cout << "Werfen Sie einige 50-Cent Stuecke ein: ";
    if( !( cin >> c50 ) || myPiggy.add50Cents( c50 ) != 0 ) break;

    cout << "Werfen Sie einige Euros ein: ";
    if( !( cin >> e1 ) || myPiggy.add1Euros( e1 ) != 0 ) break;

    cout << "Vielen Dank!" << endl;
}
//...
```

## Aufg. 4.03: OOP, Definition von Klassen Lsg. (7)

```
if( myPiggy.isFull() )
    cout << "\nDas Sparschwein ist voll!" << endl;

cout << "Das Sparschwein wird geschlachtet!" << endl;

unsigned long cents = myPiggy.breakInto( c1, c10, c50, e1 );

cout << "Im Sparschwein waren: " << endl;

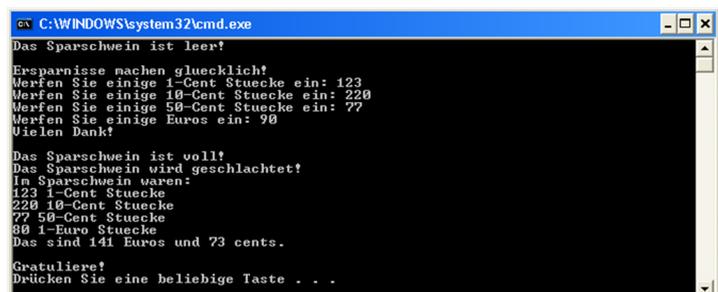
showCoins( c1, c10, c50, e1 );

cout << "Das sind " << cents / 100 << " Euros und "
    << cents % 100 << " cents."
    << "\n\nGratuliere!" << endl;

return 0;
}
```

Sparschwein ist voll?

Sparschwein wird aufge-  
brochen und gesparte  
Summe wird berechnet



```
C:\WINDOWS\system32\cmd.exe
Das Sparschwein ist leer!
Ersparnisse machen gluecklich!
Werfen Sie einige 1-Cent Stuecke ein: 123
Werfen Sie einige 10-Cent Stuecke ein: 220
Werfen Sie einige 50-Cent Stuecke ein: 77
Werfen Sie einige Euros ein: 90
Vielen Dank!
Das Sparschwein ist voll!
Das Sparschwein wird geschlachtet!
Im Sparschwein waren:
123 1-Cent Stuecke
220 10-Cent Stuecke
77 50-Cent Stuecke
90 1-Euro Stuecke
Das sind 141 Euros und 73 cents.
Gratuliere!
Drücken Sie eine beliebige Taste . . .
```

## Aufg. 4.04: OOP, Fehlersuche Lsg. (1)

a)

```
class A {  
    private:  
        long secretKey;  
    public:  
        encode( const string& );  
        decode( const string& );  
        //...  
};
```

Die Methoden `encode ( )` und `decode ( )` müssen einen Return-Typ aufweisen

b)

```
class B {  
    long numerator, denominator;  
    private:  
        void convert( double );  
        long gcd( void );  
        //...  
}
```

In der Klassendefinition fehlt die öffentliche Schnittstelle

Ein Semikolon hinter der schließenden Klammer `}` fehlt

## Aufg. 4.04: OOP, Fehlersuche Lsg. (2)

c)

```
class KFZ {  
    string hers;  
    public:  
        ~KFZ ();  
        ~KFZ( string& );  
        //...  
};
```

Der Destruktor einer Klasse hat keine Parameter und kann deshalb nicht überladen werden