Zu Übung04, Besprechung: Do., 18.06.2015 14^{00} – Neue Chemie

Aufgabe 4.01: Verständnisfragen

a) Der Operator **`new`** erwartet als Operand
   i. den Namen des Objekts, das dynamisch erzeugt werden soll.
   ii. den Typ des anzulegenden Objekts.
   iii. die Größe des Objekts in Anzahl Bytes.

b) Mit **`new`** reservierter Speicher wird mit dem Operator _____ 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.
   ii. automatisch freigegeben, wenn er nicht mehr verwendet wird.

d) Um den kontrollierten Zugriff auf die Datenelemente einer Klasse sicherzustellen, werden die Datenelemente normalerweise als **`private`** deklariert und durch _____ & _____ gelesen oder geändert.

e) Innerhalb einer Methode der Klasse können folgende Elemente dieser Klasse direkt mit ihrem Namen angesprochen werden:
   i. alle Elemente
   ii. nur die Datenelemente
   iii. nur die **`public`**-Elemente

f) Um beim Aufruf einer Methode den für den Hin- und Rücksprung notwendigen Overhead zu vermeiden, kann eine Methode als _____ definiert werden.

g) Wenn zwei verschiedene Objekte derselben Klasse angelegt werden, wird der Maschinencode für jede Methode auch zweimal erzeugt. Richtig / Falsch

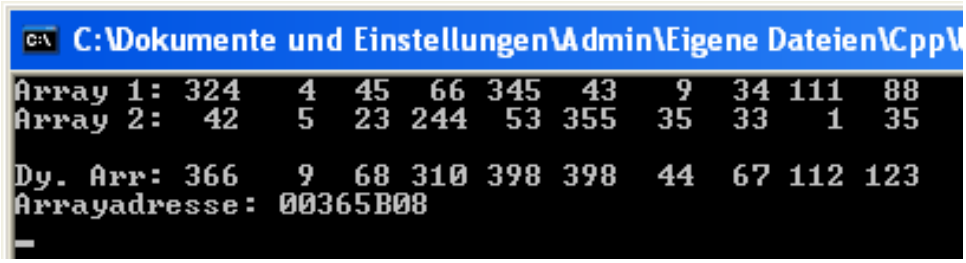
h) Bei einem Funktionsaufruf kann der durch das Erzeugen und Zerstören von Objekten bedingte Overhead vermieden werden, wenn Argumente per _____ oder _____ übergeben werden.

Aufgabe 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:



Aufgabe 4.03: OOP, Definition von Klassen

Definieren Sie eine Klasse **`PiggyBank`** zur Verwaltung der Münzen in einem Sparschwein. Die Klasse besitzt folgende von außen unsichtbare Datenelemente:

- Zählervariablen für vier Arten von Münzen (Anzahl 1-Cent, 10-Cent, 50-Cent und 1-Euro Münzen).
- Eine Variable zur Speicherung der maximalen Anzahl an Münzen, die in das Sparschwein passen.
- Ein Flag, um anzuzeigen, dass das Sparschwein aufgebrochen wurde.

Des Weiteren soll die Klasse folgende öffentliche Methoden enthalten:

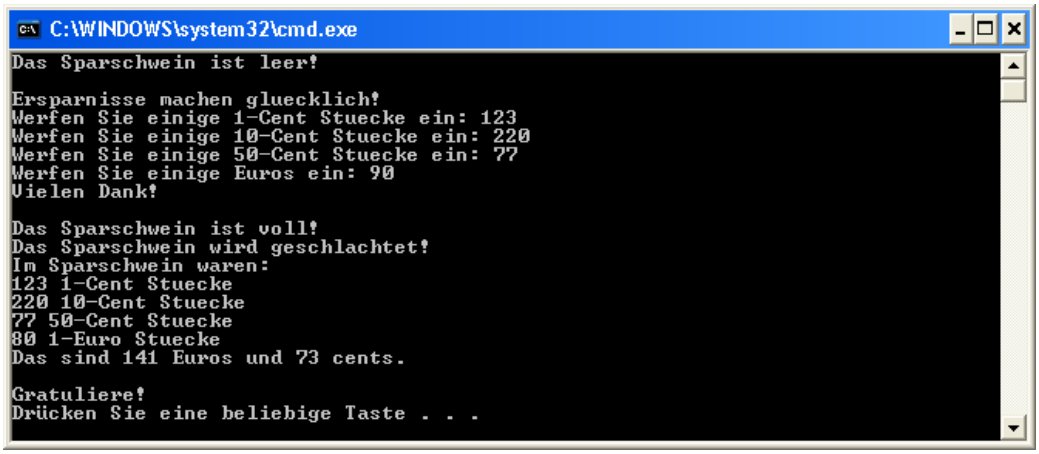
- **`PiggyBank()`** Konstruktor: Beim Aufruf des Konstruktors werden die maximale Anzahl von Münzen, die in das Sparschwein passen, und die weiteren Datenelemente zur Darstellung eines leeren Sparschweins initialisiert. Der Konstruktor soll implizit inline deklariert werden.
- **`~PiggyBank()`** Dummy-Destruktor: keine Funktion (auch implizit inline).
- **`isEmpty()`**: liefert *true*, wenn das Sparschwein leer ist, sonst *false*. Es handelt sich hier um eine explizite inline Methode.
- **`isFull()`:** liefert *true*, wenn das Sparschwein voll ist, sonst *false*. Es handelt sich hier auch um eine explizite inline Methode.
- **`isBroken()`:** liefert *true*, wenn das Sparschwein aufgebrochen ist, sonst *false*. Es handelt sich hier auch um eine explizite inline Methode.

- **`add1Cents():`** "wirft" eine übergebene Anzahl von 1-Cent-Münzen in das Sparschwein und liefert den Return-Wert 0, falls alle Münzen in das Sparschwein passen. Wenn das Sparschwein "überläuft", liefert die Methode die Anzahl der Münzen, die nicht mehr ins Sparschwein passen, als Return-Wert zurück.
- **`add10Cent(), add50Cent(), add1Euros():`** analog mit dem Unterschied, dass 10-Cent- bzw. 50-Cent- bzw. 1-Euro-Münzen in das Sparschwein "geworfen" werden.
- **`breakInto():`** bricht das Sparschwein auf und liefert die gefundenen 1-Cent-, 10-Cent-, 50-Cent- und 1-Euro-Münzen mithilfe von Referenzparametern zurück. Der Zähler der Geldstücke wird auf 0 zurückgesetzt. Der Return-Wert ist der angesparte Geldbetrag in Cents.

Hinweis: Geben Sie die Definition der Klasse **`PiggyBank`** sowie die Definitionen der inline Methoden in der Header-Datei "`PiggyBank.h`" an. In einer separaten Quelldatei "`PiggyBank.cpp`" sollen die restlichen Methoden implementiert werden.

Als letztes testen Sie die Klasse **`PiggyBank`** mit einem Anwendungsprogramm `main()` in einer separaten Quelldatei. Legen Sie ein Objekt der Klasse **`PiggyBank`** an, das bis zu 500 Münzen speichern kann. "Werfen" Sie anschließend verschiedene Münzen in das Sparschwein, bis es voll ist. Brechen Sie dann das Sparschwein auf und zeigen Sie den gesparten Betrag auf dem Bildschirm an.

Beispielausgabe des Anwendungsprogramms PiggyBank:



Aufgabe 4.04: OOP, Klassendefinition

Welche Fehler liegen in den folgenden Quellcodeabschnitten vor?

a)

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

b)

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

c)

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