

Aufgabe 1: Reihenfolge von Konstruktoraufrufen

Gehe von dem folgenden Programm aus:

```
class AKlasse
{ public:
    AKlasse()
        {cout << "AKlasse::Standardkonstruktor" << endl;}
    ~AKlasse() {cout << "AKlasse::Destruktor" << endl;}
};

class BKlasse : public AKlasse
{ public:
    BKlasse(): AKlasse()
        {cout << "BKlasse::Standardkonstruktor" << endl;}
    ~BKlasse() {cout << "BKlasse::Destruktor" << endl;}
};

class CKlasse
{ public:
    CKlasse()
        {cout << "CKlasse::Standardkonstruktor" << endl;}
    ~CKlasse() {cout << "CKlasse::Destruktor" << endl;}
};

int main(void)
{ cout << endl;
  cout << "Start von main()" << endl;
  cout << -----;
  AKlasse a_obj;    //in Teilaufgabe 2,3,4 auskommentieren
  cout << -----;
  BKlasse b_obj;
  cout << -----;
  CKlasse c_obj    //in Teilaufgabe 2,3,4 auskommentieren
  cout << -----;
  cout << "Ende von main()" << endl;
}
```

Übungstermin

a) Lokale Objekte

Rufe dieses Programm auf und beobachte die Reihenfolge der Konstruktor- und Destruktoraufrufe.

Reihenfolge:

Setze die Objektdefinition **BKlasse b_obj** in geschweifte Klammern und erkläre die geänderte Bildschirmausgabe.

Geänderte Reihenfolge und Begründung:

b) Objekte als Elemente anderer Objekte

Kommentiere zuerst die Deklarationen bzw. Definition der Objekte **a_obj** und **c_obj** in der Funktion **main** aus und füge in die Klasse **BKlasse** ein Attribut des Typs **CKlasse** ein. Dieses Attribut soll das Schutzattribut **private** haben und den Namen **c1_obj_in_BKlasse** tragen. Beachte dabei die Reihenfolge der Klassendefinitionen, die eingehalten werden muss. Beobachte in einem weiteren Probelauf den Zeitpunkt des Konstruktoraufrufs für dieses neue Attribut.

Reihenfolge:

c) Statische Objekt-Member

Ändere jetzt die Deklaration des Objekts **c1_obj_in_BKlasse** und deklariere dieses Objekt als statisches Element. Starte das Programm erneut. Was hat sich geändert und warum?

Achtung: Löschen des Bildschirms vor dem Start des Programms.

Geänderte Reihenfolge mit Begründung:

d) Komplexe Konstruktor-Destruktor-Aufrufkette

Füge erneut in die Klasse **BKlasse** ein weiteres Attribut vom Typ **CKlasse** ein. Gib diesem Attribut den Namen **c2_obj_in_BKlasse**. Analysiere die Ausgabe des Programms. (Achtung: Löschen des Bildschirms vor dem Start des Programms.)

Neue Reihenfolge mit Begründung:

Nach einer erfolgreichen Analyse dieser neuen Situation werden zusätzlich die Objekte **a_obj** und **c_obj** im **main** aktiviert (Kommentare entfernen). Die Analyse wird zwar dadurch noch komplexer, aber mit den bereits gewonnenen Erfahrungen ist es zu schaffen.

Neue Reihenfolge mit Begründung:

Aufgabe 2: Polymorphie

Programmerstellung ohne die Vorteile der Polymorphie zu nutzen (siehe Vorlesung)

Vorbereitung

a) Gegeben ist ein Klassensystem bestehend aus der Klasse **Automobil**, **Lkw** und **Omnibus**. Ergänze das Klassensystem durch eine Ausgabefunktion **printData(Automobil* a)**.

```
enum autoTyp {automobil,lkw,omnibus};

class Automobil
{ int leistung;
public:
  Automobil(int p=0){leistung=p;typ=automobil;}
  void printAutomobil(){cout << leistung << " PS";}
  autoTyp typ;
};

class Lkw: public Automobil
{ int ladeflaeche;
public:
  Lkw(int p=0){ladeflaeche=p;typ=lkw;}
  void printLkw(){cout << ladeflaeche << " qm";}
};

class Omnibus: public Automobil
{ int personen;
public:
  Omnibus(int p=0){personen=p;typ=omnibus;}
  void printOmnibus(){cout << personen << " Personen";}
};

void printData(Automobil* autoZeig)
{
}
}
```

b) Ergänze das bisherige Klassensystem durch eine von der Klasse **Automobil** abgeleitete Klasse **FormelOne**. Die Klasse **FormelOne** enthält als Attribut (entsprechend der anderen Klassen) die Höchstgeschwindigkeit **vMax**.

Gib an, an welchen Stellen des bisherigen Klassensystems Änderungen gemacht werden müssen:

Übungstermin

c) Schreibe eine **main**-Funktion, in der zu jeder der beteiligten Klassen ein Objekt erzeugt wird. Weiterhin soll für jedes dieser Objekte die Druckfunktion **printData** aufgerufen werden.

```
int main(void
{

}
}
```

Programmerstellung mit dem Vorteile der Polymorphie

Vorbereitung

d) Schreibe das Klassensystem unter a) um und setze dabei das Prinzip des dynamischen Bindens ein. An welchen Stellen der Klassendeklarationen sind Änderungen durchzuführen? Wie verändert sich vor allem die Ausgabefunktion **printData**?

e) Ergänze jetzt d) durch die Klasse **FormelOne**. Stelle auch fest, an welchen Stellen von d) der Programmcode geändert werden muss.



Übungstermin

f) Schreibe eine **main**-Funktion zum Testen von d) und e).

```
int main(void)
{

}
}
```