Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Verfasser:
D. Gachet / HTA-FR - Telekommunikation

HTA-FR

Embedded Systems 1 und 2

a.16 – Objektorientierte Programmierung in C

Klassen T-2/I-2 // 2018-2019

Inhalt

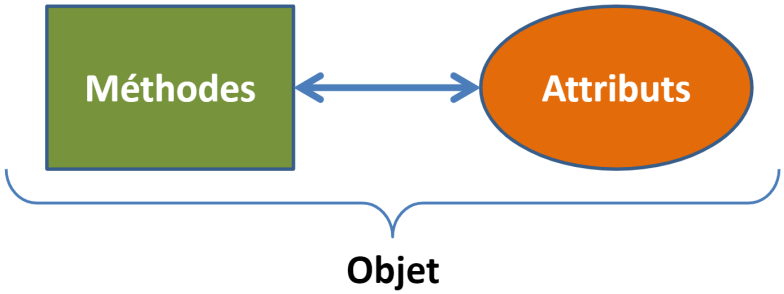
- Einführung
- Konzepte
- Vererbung
- Relationen

- **Die Programmiersprache C unterstützt nicht das Konzept von objektorientierter Programmierung. Jedoch ist es durchaus möglich, C mit objektorientiertem Stil zu programmieren. Dafür bezieht man sich auf das Grundkonzept von objektorientierter Programmierung.**

- **Was ist ein Objekt in einer objektorientierter Sprache:**

  - Attribute (Datenstruktur), welche den Zustand des Objektes definieren

  - Methoden (Algorithmen), welche das Verhalten des Objektes definieren



Deklaration

- **Um ein Objekt in C zu definieren/darzustellen reicht es:**
  - Alle Attribute, welche den Zustand des Objektes beschreiben in einer Struktur zusammenfassen
  - Methoden, welche mit dem Objekt agieren in der gleichen Struktur wie die Daten (Funktionszeiger) zusammenfassen
  - Die Referenz zu dieser Struktur als Parameter jeder Methode hinzufügen (der Zeiger auf die Struktur / das Objekt)

```
struct my_first_c_class {
   void (*m_method_1) (struct my_first_c_class*, int);
   int  (*m_method_2) (struct my_first_c_class*);
   int m_attribute_1;
   int m_attribute_2;
};
```

  - Globale Methoden können als Schnittstelle verwendet werden, um mit Objekten zu agieren. Für diese Methoden muss die Referenz auf die Struktur ebenfalls als Argument mitgegeben werden.

Definition

- Die Methoden, welche auf die Daten eines Objektes zugreifen, werden auf die gleiche Weise wie die klassischen Abläufe in C implementiert. zB:

```
void method_1(struct my_first_c_class* oref, int arg)
{
   oref->m_attribute_1 = arg + oref->m_attribute_2;
}


int method_2(struct my_first_c_class* oref)
{
   return oref->m_attribute_1;
}
```

- Diese Methoden können auch im Header-File deklariert werden, damit die abgeleiteten Klassen diese benutzen können.

- **Es ist offensichtlich, dass in der Programmiersprache C das Konzept von Konstruktor und Destruktor nicht existiert. Um diesen Mangel auszugleichen, werden eine Funktionen implementiert, welche diese Simulieren.**

- **zB die Implementation des Konstruktors:**
  - Header-file

```
extern void init_my_first_c_class (my_first_c_class* oref);
```

  - Implementation-file

```
void init_my_first_c_class (my_first_c_class* oref)
{
  oref->m_method_1 = method_1;
  oref->m_method_2 = method_2;
  oref->m_attribute_1 = 0;
  oref->m_attribute_2 = 20;
}
```

Verwendung

- **Dynamisch**
  - Deklaration des Objektes

```
struct my_first_c_class* object;
```

  - Instanziierung und Initialisierung des Objektes

```
object = malloc (sizeof(*object));
init_my_first_c_class (object);
```

  - Verwendung des Objektes

```
object->m_method_1(object, 100);
int val = object->m_method_2(object);
```

- **Statisch**
  - Deklaration des Objektes und Instanziierung

```
struct my_first_c_class object;
```

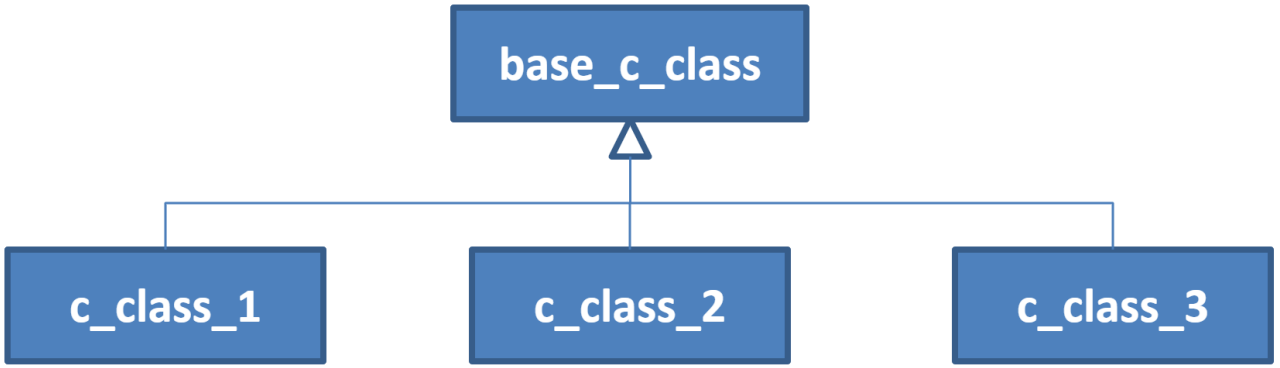
  - Initialisierung des Objektes

```
init_my_first_c_class (&object);
```

  - Verwendung des Objektes

```
object.m_method_1(&object, 100);
int val = object.m_method_2(&object);
```

- Beim der Objektorientierung ist die Konzept von Vererbung ein wesentliches Element.



- In C wird dies durch hinzufügen (Komposition) der Basisklasse in abgeleiteten Klassen realisiert. Genauer wird die Basisstruktur in die abgeleiteten Strukturen hinzugefügt.

Vererbung - Deklaration

- **Deklaration der Basisklasse**

```
struct base_c_class {
    int (*m_method_1) (struct base_c_class*, int);
    int (*m_method_2) (struct base_c_class*, int, int);
    int m_attribute_1;
    int m_attribute_2;
};
```

- **Deklaration der abgeleiteten Klasse**

```
struct c_class_1 {
    int (*m_method_3) (struct c_class_1*, int);
    struct base_c_class m_base; /* → dérivation */
    int m_attribute_3;
};
```

  - Achtung: m_base darf nicht der Zeiger auf seine Struktur sein.

Vererbung – Initialisierung

- **Initialisierung der Basisklasse**

```
void init_base_c_class (struct base_c_class* oref)       {
  oref->m_method_1 = bc_method_1;
  oref->m_method_1 = bc_method_2;
  oref->m_attribute_1 = 0;
  oref->m_attribute_2 = 0;
}
```

- **Initialisierung der abgeleiteten Klasse**

```
void init_c_class_1 (struct c_class_1* oref)
 {
  oref->m_method_3 = c1_method_3;
  init_base_c_class (&oref->m_base);
  oref->m_base.m_method_1 = c1_method_1; /* → overloading */
  oref->m_attribut_3 = 0;
}
```

Vererbung – Implementierung einer polyphonen Methode

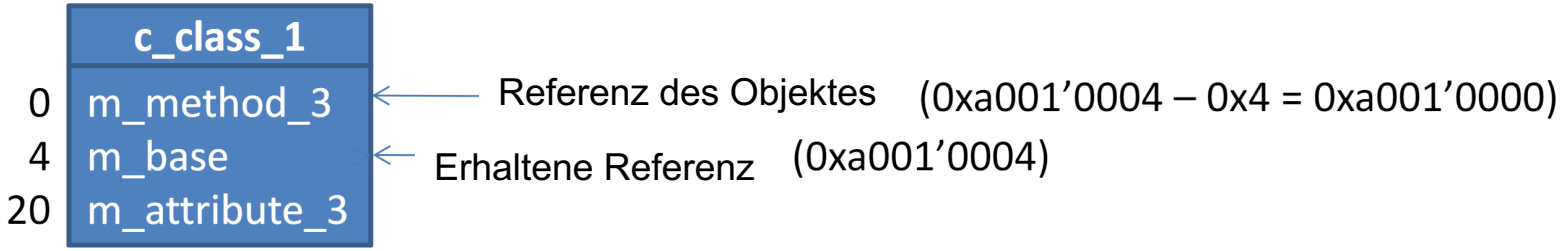
- Implementierung der Methode 1 der Basisklasse

```
int bc_method_1 (struct base_c_class* oref, int arg)
{
  return oref->m_attribute_1 + arg;
}
```

- Implementierung der Methode 1 der abgeleiteten Klasse

```
int c1_method_1 (struct base_c_class* oref, int arg)
{
   struct c_class_1* c1_ref =
     container_of(oref, struct c_class_1, m_base);
   return c1_ref->m_attribute_3 + arg;
}
```

- **Die Referenz auf ein Objekt einer abgeleiteten Klasse kann mit Hilfe eines Attributes der als Referenz gegebener Klasse berechnet werden. Dafür reicht es, das Offset dieses Attributes von der erhaltenen Referenz zu subtrahieren.**



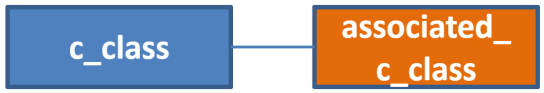
- **Diese Berechnung kann mit folgendem Makro berechnet werden:**

```
#define container_of(ptr, type, member)\
((type*) ((char*)(ptr) - offset_of(type, member)))

#define offset_of(type, member)\
((char*)&(((type*)0)->member))
```

- **Die 3 verschiedenen Typen von Relationen (Assoziation, Aggregation, Komposition) können folgendermassen in C realisiert werden:**

  - Assoziation

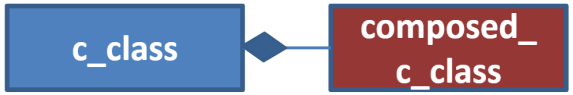



```
struct c_class {
  struct associated_c_class* m_association;
  /* ... */
};
```

  - Aggregation

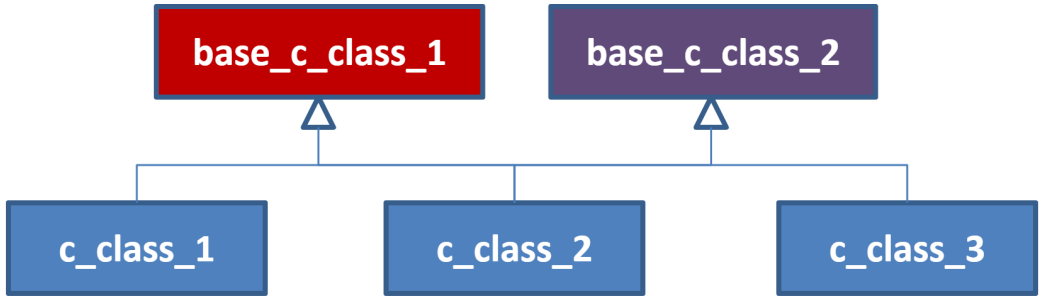

```
struct c_class {
  struct aggregated_c_class* m_aggregation;
  /* ... */
};
```

  - Komposition




```
struct c_class {
  struct composed_c_class m_composition;
  /* ... */
};
```

- **Die Technik der einfachen Vererbung kann ausgebreitet werden um die Mehrfachvererbung zu ermöglichen.**




```
struct c_class_1 {
    /* additional methods…. */

    struct base_c_class_1 m_base_1;
    struct base_c_class_2 m_base_2;

    /* additional member attributes */
};
```