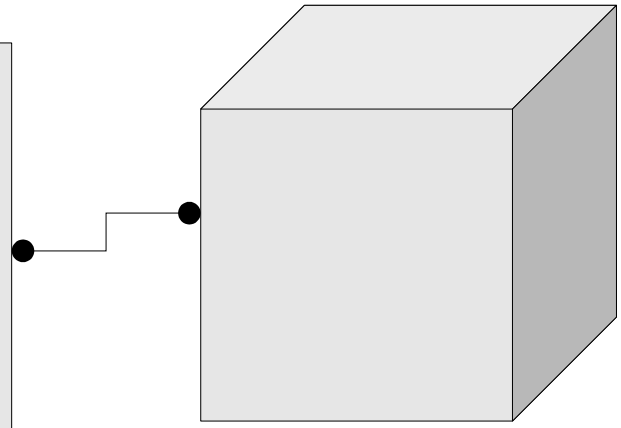
C++ - Objektorientierte Programmierung

Konstruktoren und Destruktoren

… hat eine Kantenlänge
… hat eine Füllfarbe

Kantenlänge setzen
Füllfarbe lesen
Volumen berechnen



Lebenszyklus eines Objekts

… erzeugen:
Aufruf eines Konstruktors

```
class clsQueue schlangeStandard;
```

… arbeiten:
Methoden aufrufen

```
schlangeStandard.push(1);
schlangeStandard.pop();
```

… zerstören:
Aufruf eines Destruktors

Konstruktoren

- „Erzeuge von dem Bauplan ein Ding“.
- Erzeugung (Instanziierung) eines Objekts.
- Spezielle Methoden, die bei der Deklaration von Objekten aufgerufen werden.
- Initialisierung von Attributen für ein Objekt vom Typ „Klasse“.
- Bereitstellung von Ressourcen etc., die von dem Objekt benötigt werden.

… in einer Klasse definieren (inline)

```
clsWuerfel() {
    kantenlaenge = 1;
}

clsWuerfel(int laenge)
{
    kantenlaenge = laenge;
    farbe = 'b';
}

clsWuerfel(int laenge, char farbe)
        :kantenlaenge(laenge), farbe(farbe)
{}
```

Kopf und Rumpf eines Konstruktors

clsWuerfel(int laenge)	Kopf: Wie wird das Objekt instanziiert?

{
kantenlaenge = laenge;
farbe = 'b';
}
	Rumpf:
	Mit welchen Werten werden die Attribute initialisiert?

Kopf eines Konstruktors

konstruktorname	(	parameterliste	)	;

- Der Konstruktor-Name entspricht immer dem Klassennamen.
- In runden Klammern folgt dem Namen eine Parameterliste. Die Liste kann leer sein.

Rumpf

- Beginn und Ende mit den geschweiften Klammern.
- Initialisierung von Attributen in der Klasse.
- Ein Konstruktor gibt keinen Wert zurück. Das Schlüsselwort `return` ist in einem Konstruktor nicht erlaubt.

Standard-Konstruktor

```
clsWuerfel::clsWuerfel() {
    kantenlaenge = 1;
}
```

Beispiele/cppOOP_003_clsWuerfel...

- Die Attribute werden mit Hilfe von Standardwerten initialisiert.
- Pro Klasse kann nur ein Standard-Konstruktor deklariert werden.
- Falls kein Konstruktor in der Klasse deklariert ist, wird ein Standard-Konstruktor automatisch vom Compiler erzeugt.

… in einer Klasse definieren (inline)

```
class clsWuerfel {
   public:
          clsWuerfel() {
                 kantenlaenge = 1;
          }
};
```

- Instanziierung von Objekten mit Hilfe von Konstruktoren.
- Konstruktoren müssen immer öffentlich definiert werden.

Deklaration als Prototyp in einer Header-Datei

konstruktorname	(		)	;
clsWuerfel	(		)	;

- Der Kopf des Konstruktors wird als Prototyp genutzt.
- Eine Parameterliste ist nicht vorhanden. Die runden Klammern sind leer.
- Die Anweisung wird mit einem Semikolon abgeschlossen.

Definition in einer Quelldatei

```
clsWuerfel::clsWuerfel() {
    kantenlaenge = 1;
}
```

- Der Kopf des Konstruktors (clsWuerfel()) entspricht dem Prototypen in der Klasse.
- Der Bereichsoperator (Scope-Operator) wird aus zwei aufeinander folgenden Doppelpunkten gebildet. Der Operator verbindet den Kopf mit der Klasse, in deren Bereich der Konstruktor deklariert ist.
- Im Rumpf werden die Attribute auf Werte, die am häufigsten vorkommen, gesetzt.

Instanziierung mit einem Standard-Konstruktor

class	klassenname	variablenname	;
class	clsWuerfel	myWuerfel	;

- Dem Namen der Objektvariablen folgen keine runde Klammern.
- Durch den angegebenen Klassennamen wird automatisiert der dazugehörige Standard-Konstruktor aufgerufen.

Hinweise

```
class clsWuerfel myWuerfel;
```

Instanziierung mit Hilfe des Standard-Konstruktors.

```
class clsWuerfel myWuerfel();
```

Aufruf der Funktion `myWuerfel()`. Es wird vermutet, dass die Funktion den Rückgabetyp `clsWuerfel` hat. Der Fehler „error: request for member '...' in '...', which is of non-class type '...'“ wird ausgegeben.

Regeln

- In einem Konstruktor kann kein Objekt von der Klasse erzeugt werden, in der dieser deklariert ist.
- Bei der Erzeugung eines Objekts wird von außen auf ein Konstruktor zugegriffen. Konstruktoren haben ein öffentliches Zugriffsrecht.
- Konstruktoren geben an den Erzeuger des Objektes keine Werte zurück.

Allgemeine Konstruktoren

```
clsWuerfel(int laenge)
{
    kantenlaenge = laenge;
    farbe = 'b';
}
```

Beispiele/cppOOP_003_clsWuerfel...

- Attribut-Werte werden mit Hilfe der übergebenen Parameter initialisiert.
- In einer Klasse können beliebig viele allgemeine Konstruktoren vorkommen.

… in einer Klasse definieren (inline)

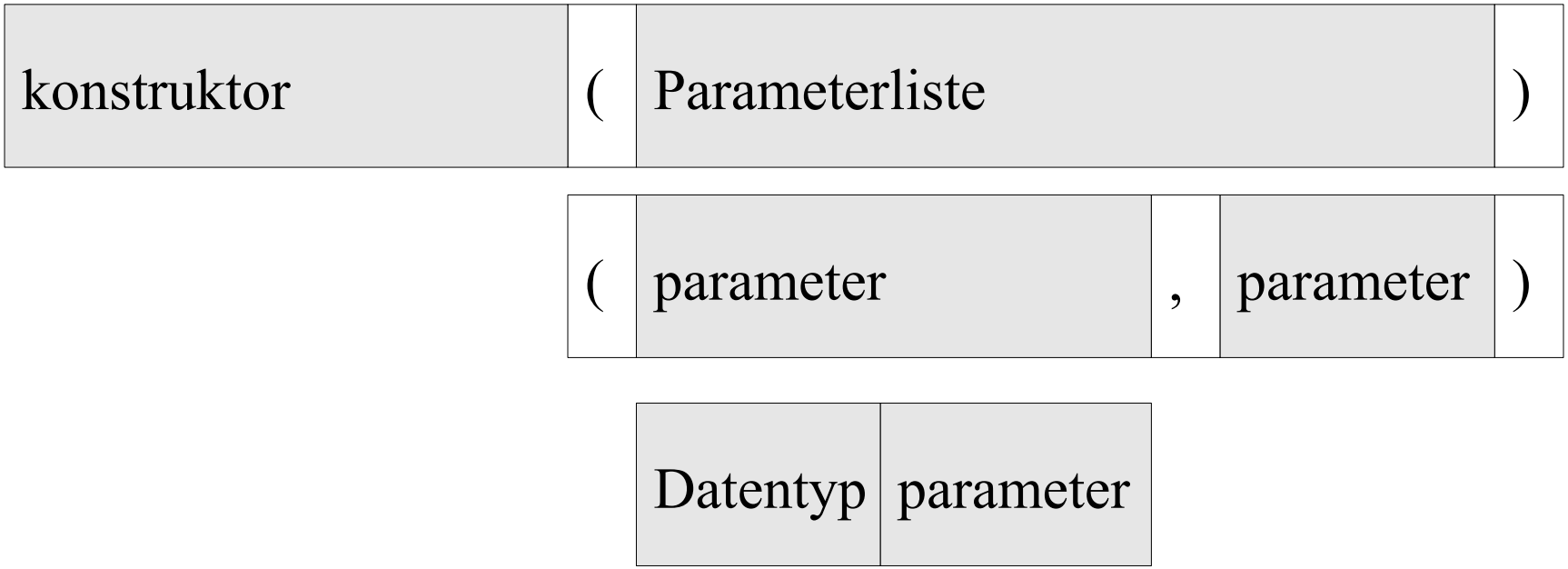
```
class clsWuerfel {

   public:
          clsWuerfel(int laenge)
          {
             kantenlaenge = laenge;
             farbe = 'b';
          }
};
```

Deklaration als Prototyp in einer Header-Datei

konstruktorname	(	parameterliste	)	;
clsWuerfel	(	double laenge, char farbe	)	;

- Der Kopf des Konstruktors wird als Prototyp genutzt.
- Dem Namen des Konstruktors folgen die runden Klammern. In den runden Klammern befindet sich eine Liste von Parametern.
- Die Parameter werden bei der Instanziierung übergeben.
- Die Anweisung wird mit einem Semikolon abgeschlossen.

Liste von Parametern



- Die Liste von Parametern setzt sich aus x beliebigen Parametern, getrennt durch ein Kommata zusammen.
- Für jeden Parameter wird ein Datentyp und ein eindeutiger Name in der Liste angegeben.

Definition in einer Quelldatei

```
clsWuerfel::clsWuerfel(int laenge)
{
    kantenlaenge = laenge;
    farbe = 'b';
}
```

- Der Kopf des Konstruktors in der Quelldatei entspricht dem Prototypen in der Klasse.
- Im Rumpf werden die Attribut-Werte mit Hilfe der Parameter gesetzt.

Kopf des Konstruktors

clsWuerfel	::	clsWuerfel	(	int laenge	)
Klasse	::	klassename	(	Parameterliste	)

- Links vom Bereichsoperator: In welcher Klasse ist der Konstruktor deklariert?
- Rechts vom Bereichsoperator: Von welcher Klasse werden Objekte erzeugt?
- In den runden Klammern: Welche Parameter werden den Konstruktor übergeben?

Instanziierung mit einem allgemeinen Konstruktor

class	klassenname	variablenname	(	parameterliste	)	;
class	clsWuerfel	blueWuerfel	(	2	)	;
class	clsWuerfel	redWuerfel	(	3, 'r'	)	;

- Dem Namen der Objektvariablen folgen runde Klammern.
- In Abhängigkeit der Liste der Parameter in den runden Klammern wird der Konstruktor aufgerufen.

Beispiel

```
class clsWuerfel blueWuerfel(2);
```

```
clsWuerfel::clsWuerfel(int laenge)
{
   kantenlaenge = laenge;
   farbe = 'b';
}
```

```
class clsWuerfel redWuerfel(3, 'r');
```

```
clsWuerfel::clsWuerfel(int laenge,
                       char farbe)
{
   kantenlaenge = laenge;
   this->farbe = farbe;
}
```

Überladung von Konstruktoren

- Alle Konstruktoren haben den gleichen Namen.
- Aber jeder Konstruktor unterscheidet sich in der Parameterliste.
- Mit Hilfe der Anzahl der Parameter und / oder dem unterschiedlichen Datentyp der Parameter wird dem Aufruf der passende Konstruktor zugeordnet.

Sichtbarkeit von Attributen

- Wann ist ein Zugriff auf ein Attribut möglich?
- In welchen Bereich kann der Inhalt / der Wert des Attributs gelesen und verändert werden?
- Parameter oder lokale Variablen in Methoden und Konstruktoren können Attribute gleichen Namens überdecken.

Attribute

```
clsWuerfel::clsWuerfel(int laenge, char farbe)
{
   kantenlaenge = laenge;
   this->farbe = farbe;
}
```

- Attribute des aufrufenden Objekts werden gelesen oder geändert.
- Die Attribut-Werte des zu instanziierenden Objekts werden initialisiert.

Parameter

```
clsWuerfel::clsWuerfel(int laenge, char farbe)
{
   kantenlaenge = laenge;
   this->farbe = farbe;
}
```

- Parameter sind lokal. Die Parameter können nur in dem Rumpf der Methode oder Konstruktor genutzt werden, zu dem der Kopf gehört.

Überdeckung

```
clsWuerfel::clsWuerfel(int laenge, char farbe)
{
   kantenlaenge = laenge;
   this->farbe = farbe;
}
```

- Parameter oder lokale Variablen in einem Anweisungsblock können Attribute einer Klasse gleichen Namens überdecken.
- Ein überdecktes Attribut kann mit Hilfe des Zeigers this gesetzt oder gelesen werden.

this-Zeiger

this	->	attribut
this	->	farbe

- Das Schlüsselwort this ist ein Platzhalter für das aktuelle Objekt.
- Synonym für das Objekt, welches mit Hilfe des aufgerufenen Konstruktors erzeugt wird.
- Es werden die Instanzvariablen des Aufrufers verändert.

Default-Werte nutzen

```
clsWuerfel(int, char farbe = 'b');
```

```
clsWuerfel::clsWuerfel(int laenge, char farbe)
{
   kantenlaenge = laenge;
   this->farbe = farbe;
}
```

Beispiele/cppOOP_003_clsWuerfel...

Erläuterung

- Dem Default-Parameter folgen nur weitere Default-Parameter. Default-Parameter stehen am Ende der Parameter-Liste.
- In der Deklaration eines Konstruktors wird der Standardwert angegeben. In der Definition des Konstruktors wird nur der Kopf ohne Angabe des Standardwertes genutzt.

Andere Möglichkeit „Überladung“

```
clsWuerfel(int laenge)
{
    kantenlaenge = laenge;
    farbe = 'b';
}
```

```
clsWuerfel::clsWuerfel(int laenge,
                       char farbe)
{
    kantenlaenge = laenge;
    this->farbe = farbe;
}
```

Beispiele/cppOOP_003_clsWuerfel...

Instanziierung

```
class clsWuerfel blueWuerfel(2);
class clsWuerfel redWuerfel(3, 'r');
```

- In Abhängigkeit der Parameterliste wird der entsprechende Konstruktor aufgerufen.

Initialisierungslisten von Konstruktoren

- Attribute können mit Hilfe von sogenannten Initialisierungslisten auf einen Anfangswert gesetzt werden.
- Initialisierungslisten werden nur bei der Definition eines Konstruktors angegeben.
- Bei einem Aufruf eines Konstruktors werden zuerst die Attribut deklariert und dann mit Hilfe der Initialisierungslisten auf einen Anfangswert gesetzt. Anschließend werden die Anweisungen im Rumpf ausgeführt.

Beispiel

```
clsWuerfel(int, char);
```

```
clsWuerfel::clsWuerfel(int laenge, char farbe)
                        :kantenlaenge(laenge), farbe(farbe)
{

}
```

Beispiele/cppOOP_003_clsWuerfel...

Initialisierungsliste im Kopf

clsWuerfel	::	clsWuerfel	(		)	:	kantenlaenge(laenge)
scope	::	konstruktor	(	parameterliste	)	:	initialisierungsliste

Kopf des Konstruktors

Initialisierungsliste

Aufbau der Initialisierungsliste

kopf	:	attribut	(	wert	)	,	attribut	(	wert	)

- Die einzelnen Elemente in der Liste werden durch Kommata getrennt.
- Die Reihenfolge der Attribute in der Initialisierungsliste entspricht der Reihenfolge in der Klassendeklaration.
- Dem Attribut wird in den runden Klammern ein Wert entsprechend des Datentyps übergeben.

Flache Kopie

- Der Compiler stellt einen Kopierkonstruktor und den Zuweisungsoperator zur Verfügung.
- Die Inhalte der Instanzvariablen des Quellobjektes werden 1:1 in das Zielobjekt kopiert.
- Bei der Nutzung von Zeigern führt diese Art des Kopierens zu Fehlern.

Tiefe Kopie

- Der Entwickler stellt einen Kopierkonstruktor zur Verfügung. Der Zuweisungsoperator wird überladen.
- Die Inhalte der Instanzvariablen des Quellobjektes werden 1:1 in das Zielobjekt kopiert.
- Der Wert, auf den Zeiger verweisen, wird kopiert.

Kopierkonstruktor

```
clsQueue(const clsQueue& orig);
```

```
clsWuerfel::clsWuerfel(const clsWuerfel& orig) {
    this->kantenlaenge = orig.kantenlaenge;
    this->farbe = orig.farbe;
}
```

Beispiele/cppOOP_003_clsWuerfel...

Instanziierung mit Hilfe des Kopierkonstruktors

```
class clsWuerfel wuerfelKopie(redWuerfel);
```

- In den runden Klammern wird ein Objekt übergeben.
- Die Attribut-Werte des Quellobjekts redWuerfel werden den Attributen des Objekts wuerfelKopie zugewiesen.

Hinweis

```
class clsWuerfel redWuerfel(3, 'r');
class clsWuerfel wuerfelKopie(redWuerfel);
```

Instanziierung der Objekte.

```
wuerfelKopie = redWuerfel;
```

In einer Objektvariable wird die Speicher-adresse abgelegt. Mit Hilfe der Standard-zuweisung verweisen beide Variablen auf das gleiche Objekt.

Destruktoren

- Aufruf bei Zerstörung eines Objektes. Statische Objekte werden bei Verlassen des Gültigkeitsbereiches zerstört. Objekte, die mit `new` erzeugt wurden, werden durch den Befehl `delete` zerstört.
- Wenn kein Destruktor definiert ist, wird automatisiert ein Destruktor vom Compiler angelegt.
- Wenn von der Klasse Ressourcen angefordert werden, wird ein Destruktor benötigt.

… in einer Klasse definieren (inline)

```
~clsWuerfel() {

}
```

Kopf und Rumpf eines Destruktors

~clsWuerfel()	Kopf

{
 	
}
	Rumpf

Kopf eines Destruktors

~	destruktorname	(	)	;

- Der Destruktor beginnt mit dem Tilde-Zeichen.
- Der Destruktor-Name entspricht immer dem Klassennamen.
- Dem Destruktor werden keine Parameter übergeben.

Objektvariable

```
class clsWuerfel myWuerfel;
class clsWuerfel blueWuerfel(2);
class clsWuerfel redWuerfel(3, 'r');
```

- Variable vom Typ „Klasse“.
- Die Klasse muss durch die Anweisung #include eingebunden werden.
- In der Variablen wird als Wert die Speicheradresse für ein Objekt abgelegt.

Instanziierung mit Hilfe des Standard-Konstruktors

class	clsWuerfel	myWuerfel	;
class	klassenname	variablenname	;

- Variable vom Typ einer Klasse.
- Dem Variablennamen folgen keine runden Klammern.
- Das Objekt wird mit den Standardwerten initialisiert.

Hinweise

- Das Schlüsselwort class muss bei nicht eindeutigen Klassennamen gesetzt werden. Bei benutzerdefinierten Klassen sollte das Schlüsselwort aber immer an den Anfang der Deklaration gesetzt werden.
- Der Variablenname ist frei wählbar. Der Name sollte aber das Objekt widerspiegeln.
- Die Deklarationsanweisung kann überall im Anweisungsblock erfolgen. Variablen sollten aber am Anfang eines Anweisungsblockes oder kurz vor deren ersten Nutzung deklariert werden.

Instanziierung mit allgemeinen Konstruktoren

class	klassenname	variablenname	(	parameterliste	)	;
class	clsWuerfel	redWuerfel	(	3, 'r'	)	;

- In den runden Klammern werden Anfangswerte für den Bau übergeben.
- Die Attribute des Objekts werden mit Hilfe der übergebenen Werte initialisiert.

Aufruf von Methoden

push	(	1	)	;
methodenname	(	parameterliste	)	;

- Jede Methode wird mit Hilfe ihres Namens aufgerufen.
- Der Name ist ein eindeutiger Platzhalter für die Methode einer Klasse.
- Dem Namen folgt die Parameterliste. Die Parameterliste beim Aufruf entspricht der Liste im Methodenkopf. Die Liste kann leer sein.

Aufruf über eine Objektvariable

schlange	.	push	(	1	)	;
objektvariable	.	methodenname	(	parameterliste	)	;

- Die Methode, rechts vom Punktoperator, verändert das Objekt links.
- Die Objektvariable verweist auf eine Instanz vom Typ „Klasse“. In diesem Typ ist die aufzurufende Methode deklariert.

Rückgabewerte einer Methode

```
int anzahlObjekte;
anzahlObjekte = schlange.getAnzahlSchlange();
```

- Eine Methode kann einen Rückgabewert haben, muss aber nicht.
- Der Rückgabewert kann in einer Variablen gespeichert werden.
- Die Variable und der Datentyp der Methode sollten gleich sein.

Array von Objekten

```
class clsWuerfel vieleWuerfel[4];
cout << "1.  Würfel Länge: " << vieleWuerfel[0].getKantenlaenge() << '\n';
cout << "1.  Würfel Farbe " << vieleWuerfel[0].getFarbe() << '\n';

vieleWuerfel[1].setFarbe('r');
vieleWuerfel[1].setKantenlaenge(4.5);
cout << "2. Würfel Länge: " << vieleWuerfel[1].getKantenlaenge() << '\n';
cout << "2. Würfel Farbe: " << vieleWuerfel[1].getFarbe() << '\n';
```

Beispiele/cppOOP_003_clsWuerfel..

Deklaration

	int	monat	[	12	]	;
class	clsWuerfel	vieleWuerfel	[	4	]	;
Datentyp		name	[	anzahl	]	;

- Der Name eines Arrays ist frei wählbar.
- Direkt im Anschluss an den Namen folgt, in eckigen Klammern, die Anzahl der Elemente.
- Die Elemente sind vom Datentyp ...

Instanziierung der Objekte

```
class clsWuerfel vieleWuerfel[4];
```

- Jedes Element im Array enthält eine Instanz von der Klasse...
- Objekte in Arrays werden mit Hilfe des Standard-Konstruktors erzeugt.

Instanz in einem Array

wuerfel	[	0	]
array	[	index	]

- Dem Namen des Arrays folgt der Index in eckigen Klammern.
- Der Index identifiziert eindeutig ein Element in dem Array.
- Das erste Element hat den Index 0 und so weiter.
- Jedes Element in einem Array stellt ein Objekt von dem Typ „Klasse“ dar.
- In jeder Box befindet sich eine Instanz. Die Instanz ist vom Typ „Klasse“ des Arrays.

Aufruf von Methoden über ein Array-Element

wuerfel	[	0	]	.	getFarbe	(	1	)	;
array	[	index	]	.	methode	(	parameterliste	)	;

- Das Element links vom Punktoperator beantwortet folgende Fragen: In welcher Klasse ist die Methode deklariert? Von welchen Element sollen Attribut-Werte gelesen oder verändert werden?
- Das Element rechts vom Punktoperator beantwortet die Frage: Mit welchen Werkzeug wird der Attribut-Wert gelesen oder verändert?