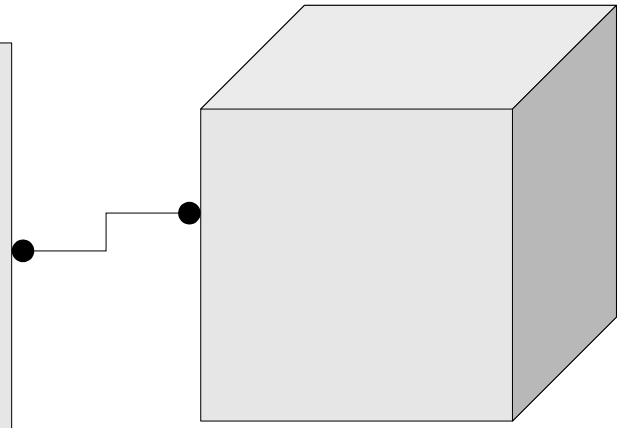
C++ - Objektorientierte Programmierung

Konstante und statische Elemente

… hat eine Kantenlänge
… hat eine Füllfarbe

Kantenlänge setzen
Füllfarbe lesen
Volumen berechnen



Konstante Attribute

- Konstante Attribute können nie im Leben eines Objektes verändert werden.
- Sobald ein Objekt erzeugt wird, muss die konstante Instanzvariable gesetzt werden.

… in C++

Beispiele/cppOOP_004_clsQueue...

```
class clsQueue {

Public:

private:
    const int anzahlElemente_max;

};
```

Deklaration

const	int	anzahl_Elemente-max	;
const	Datentyp	Name	;

- Die Deklaration von Instanzvariablen beginnt mit dem Schlüsselwort const.
- Der Datentyp legt die Art des zu speichernden Wertes und den benötigten Speicherplatz fest.
- Der Name der Konstanten ist frei wählbar.

Initialisierung von Konstanten

- Konstante Attribute können nicht gleichzeitig deklariert und initialisiert werden.
- Konstanten müssen mit Hilfe der Initialisierungsliste eines Konstruktors gesetzt werden.

Initialisierungslisten

```
clsQueue::clsQueue():anzahlElemente_max(5)
{

}
```

- Initialisierungslisten werden nur bei der Definition eines Konstruktors angegeben.
- Konstante Attribute müssen mit Hilfe von Initialisierungslisten gesetzt werden.
- Alle anderen Attribute können mit Hilfe der Initialisierungsliste gesetzt werden.

Beispiele/cppOOP_004_clsQueue...

Leibniz Universität Hannover

Initialisierungslisten

`clsQueue();`	Deklaration des Konstruktors in der Header-Datei.

`clsQueue::clsQueue():anzahlElemente_max(5)`
`{`
`  ptrKopf = new int[anzahlElemente_max];`
`  ptrEnde = ptrKopf;`
`  this->setArrayNull();`
`  anzahlElemente_aktuell = 0;`
`  anzahlSchlange++;`
`}`
	Definition des Konstruktors in der dazugehörigen Implementierungs datei.
	Dem Namen des Konstruktors folgt die Initialisierungs-liste.

Beispiele/cppOOP_004_clsQueue...

Initialisierungsliste im Kopf

clsQueue	::	clsQueue	(		)	:	anzahlElemente_max(5)
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 Attribut in der Initialisierungsliste entspricht der Reihenfolge in der Klassendeklaration.
- Dem Attribut wird in den runden Klammern ein Wert entsprechend des Datentyps übergeben.

Initialisierung der Attribute im Rumpf

```
clsQueue::clsQueue()
{
    anzahlElemente_max = 5;
    ptrKopf = new int[anzahlElemente_max];
}
```

- Die Attribute werden deklariert.
- Anschließend werden die Attribute mit einem beliebigen Wert initialisiert.
- Der Rumpf des Konstruktors wird ausgeführt. Den Attributen wird ein Wert durch die Anweisungen im Rumpf zugewiesen.

Initialisierung der Attribute mit einer Liste

```
clsQueue::clsQueue():anzahlElemente_max(5)
{
   ptrKopf = new int[anzahlElemente_max];
}
```

- Die Attribute werden deklariert.
- Anschließend wird die Initialisierungsliste ausgeführt. Die Attribute werden entsprechend der Angaben in der Liste initialisiert.
- Die Anweisungen im Rumpf werden ausgeführt.

Konstante Methoden

- Konstante Methoden lesen Attribute, verändern diese aber nicht.
- Eine konstante Methode kann im Methodenrumpf nur eine weitere konstante Methode aufrufen.
- Konstante Objekte können nur konstante Methoden der eigenen Klasse aufrufen.

… in C++

```
int peek() const;
```

```
int clsQueue::peek() const
{
   return(*ptrEnde);
}
```

Beispiele/cppOOP_004_clsQueue...

Methodenkopf

Datentyp	methodename	(	Parameterliste	)	const

- Am Ende des Methodenkopfes steht das Schlüsselwort const.
- Das Schlüsselwort wird in der Deklaration sowohl als auch in der Definition gesetzt.

Konstante Objektvariablen

- Das Objekt, auf welches verwiesen wird, ist nicht veränderbar.
- Schreibschutz für eine Instanz.
- Konstante Objekte können nur konstante Methoden aufrufen.

Beispiel

```
const class clsQueue schlangeConst(schlange);
cout << "\nKonstante Schlange: ";
schlangeConst.writeSchlange();

schlangeConst.schlangeZuLang();
schlangeConst.writeSchlange();

cout << "\n Erstes Element: " << schlangeConst.peek() << "\n";
```

Beispiele/cppOOP_004_clsQueue...

Veränderliche Attribute

mutable	int	*ptrEnde	;
mutable	Datentyp	Name	;

- Seit C++ 11.
- Die, mit mutable gekennzeichneten Attribute, lassen sich auch durch konstante Methoden verändern.

Leibniz Universität Hannover

Nutzung

```
class clsQueue {

public:
    void schlangeZuLang() const;

private:
    int *ptrKopf;
    mutable int *ptrEnde;
    mutable int anzahl_aktuell;

    const int anzahlElemente_max;
};
```

```
void clsQueue::schlangeZuLang() const{
    int * ptrTemp;
    ptrTemp = ptrKopf;

    do{
      *ptrTemp = *(ptrTemp + 1);
      ptrTemp++;
    }while(ptrTemp < ptrEnde);

    ptrEnde = (ptrEnde - 1);
    anzahl_aktuell--;
}
```

Beispiele/cppOOP_004_clsQueue

Instanzvariablen

- Attribute, die für jedes Objekt im Speicher abgelegt werden.
- Variablen einer Klasse, die bei der Instanziierung von Objekten, neu angelegt werden.
- Eigenschaften, die die Ausprägung eines Objektes beschreiben.

Klassenvariablen

- Statische Variablen.
- Variablen, die für alle Objekte vom Typ „Klasse“ den gleichen Wert haben.
- Einmalige Speicherung für alle Objekte einer Klasse.
- Aufruf über den Klassennamen oder über eine Objektvariable von dem Typ.

Deklaration

static	int	anzahlSchlange	;
static	Datentyp	Name	;

Beispiele/cppOOP_004_clsQueue...

- Klassenvariablen werden innerhalb der Klasse deklariert, aber nicht initialisiert.
- Klassenvariablen werden häufig zum Zählen von Objekten einer bestimmten Klasse genutzt.

Initialisierung

int	clsQueue	::	anzahlSchlange	=	0	;
Datentyp	scope	::	variable	=	wert	;

- Statische Attribute werden außerhalb der Klasse initialisiert.
- Häufig werden statische Attribute in der Quelldatei einer Klasse initialisiert.
- Das statische Attribut lebt so lange wie ein Objekt von der Klasse existiert.

Statische Methoden

- Statische Methoden existieren auch ohne ein Objekt von der Klasse.
- Statische Methoden können nur auf Klassenvariablen zugreifen.

… in C++

```
static int getAnzahlSchlange();
```

```
int clsQueue::getAnzahlSchlange()
{
   return anzahlSchlange;
}
```

Beispiele/cppOOP_004_clsQueue...

Methodenkopf

static	Datentyp	methodename	(	Parameterliste	)

- Am Anfang des Methodenkopfes steht das Schlüsselwort static.
- Das Schlüsselwort wird nur in der Deklaration einer Methode gesetzt.

Aufruf über eine Objektvariable

```
cout << "\n Anzahl der Schlangen: " << schlange.getAnzahlSchlange();
```

Beispiele/cppOOP_004_clsQueue...

- Statische Methode können genauso wie alle anderen Methoden über die Objektvariable aufgerufen werden.
- Die Objektvariable und die statische Methode werden mit dem Punktoperator verbunden.

Aufruf über den Klassennamen

```
cout << "\n Anzahl der Schlangen: " << clsQueue::getAnzahlSchlange();
```

Beispiele/cppOOP_004_clsQueue...

- Statische Methoden können auch direkt über den Klassennamen aufgerufen werden.
- Der Klassenname und die statische Methode werden mit Hilfe des Bereichsoperators (zwei aufeinanderfolgende Doppelpunkte) verbunden.

Zeiger als Attribute

- Verweis auf eine Speicherstelle.
- Speicherung einer Speicheradresse.
- Platzhalter für den Beginn eines Speicherbereichs, an dem ein Wert vom Datentyp ... gespeichert ist.

… in C++

Beispiele/cppOOP_004_clsQueue...

```
class clsQueue {

public:

private:
        int *ptrKopf;
        int *ptrEnde;
};
```

Deklaration

int	*ptrKopf	;
Datentyp	*name	;

- Das Sternchen kennzeichnet einen Zeiger. Um Fehler zu vermeiden, sollte das Sternchen immer direkt vor dem Namen des Zeigers geschrieben werden.
- In diesem Beispiel verweist der Zeiger auf einen Wert vom Typ int. Der Zeiger verweist auf einen Speicherbereich, der so groß ist, dass dort eine Ganzzahl vom Typ int abgelegt werden kann.

Beispiele für die Initialisierung

```
clsQueue::clsQueue(int elemente):anzahlElemente_max(elemente)
{
   ptrKopf = new int[anzahlElemente_max];
   ptrEnde = ptrKopf;
}
```

Beispiele/cppOOP_004_clsQueue...

Arbeiten mit dem Adressoperator

```
int *ptrKopf;
int varElemente;

ptrKopf = &varElemente;
```

- Die Variable varElemente ist an einer bestimmten Adresse im Speicher abgelegt.
- Mit Hilfe des Adressoperators & wird der Ablageort der Variablen in dem Zeiger gespeichert.
- Der Zeiger und der Wert, auf den der Zeiger verweist, sollten den gleichen Datentyp haben.

Zuweisung einer Adresse

```
int *ptrKopf;
int *ptrEnde

ptrEnde = ptrKopf;
```

- Die Adresse, die in der Variablen ptrKopf gespeichert ist, wird dem Zeiger ptrEnde zugewiesen.
- Beide Zeiger verweisen auf dasselbe Element.

Anforderung von Speicher

```
int *ptrKopf;
ptrKopf = new int[anzahlElemente_max];
```

- Mit Hilfe des Schlüsselwortes new wird Speicher angefordert. Der Speicher wird im Heap des Rechners bereitgestellt.
- Dem Schlüsselwort folgt die benötigte Größe des Speicherbereichs. In diesem Beispiel wird Speicher für x Elemente vom Datentyp int angefordert. Die Anfangsadresse wird in dem Zeiger gespeichert. Der Zeiger verweist auf das erste Element.
- Falls nicht genügend Speicher vorhanden ist, wird ein undefinierter Zeiger zurück geliefert.

Lebenszyklus eines Objekts

… erzeugen:
Aufruf eines Konstruktors

clsQueue(int elemente)

… arbeiten:
Methoden aufrufen

… zerstören:
Aufruf eines Destruktors

~clsQueue()

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.
- Falls kein Destruktor vorhanden ist, legt der Compiler automatisch einen an.

Nutzung

- In einem Destruktor muss dynamisch angeforderter Speicher wieder freigegeben werden.
- Datei werden geschlossen.
- Angeforderte Ressourcen werden freigegeben.

… in einer Klasse definieren (inline)

```
~clsQueue()
{
    delete[] ptrKopf;
    ptrKopf = 0;
    ptrEnde = 0;
    anzahlSchlange--;
}
```

Kopf und Rumpf eines Konstruktors

~clsQueue()	Kopf

{
delete[] ptrKopf;
ptrKopf = 0;
ptrEnde = 0;
anzahlSchlange--;
}
	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.

Prototyp und Definition

```
~clsQueue();
```

```
clsQueue::~clsQueue()
{
    delete[] ptrKopf;
    ptrKopf = 0;
    ptrEnde = 0;
    anzahlSchlange--;
}
```

Beispiele/cppOOP_004_clsQueue...

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.
- Nachteil: Wenn Zeiger kopiert werden, wird der Verweis auf das Objekt kopiert. Aber nicht das Objekt, auf welches sie verweisen.

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);
```

Beispiele/cppOOP_004_clsQuery...

- Als Parameter wird das Quellobjekt übergeben.
- Die Adresse des Quellobjektes wird als Referenz übergeben. Eine Referenz verweist auf ein konstantes Objekt. Das Objekt kann nicht im Konstruktor verändert werden.

Implementierung: Setzen der Zeiger

```
clsQueue::clsQueue(const clsQueue& orig)
    int *ptrZiel;
    int *ptrQuelle;

    this->anzahlElemente_max = orig.anzahlElemente_max;
    this->ptrKopf = new int[anzahlElemente_max];
    this->ptrEnde = ptrKopf;
    this->setArrayNull();
    this->anzahlElemente_aktuell = 0;
    anzahlSchlange++;

    ptrZiel = this->ptrKopf;
    ptrQuelle = orig.ptrKopf;
```

Implementierung: Füllen des Arrays

```
do{
   *(ptrZiel + anzahlElemente_aktuell)  =
                        *(ptrQuelle + anzahlElemente_aktuell);

    this->ptrEnde = (ptrZiel + anzahlElemente_aktuell);
   this->anzahlElemente_aktuell++;

 }while(
        (this->anzahlElemente_aktuell < orig.anzahlElemente_aktuell)
       &&
      (this->anzahlElemente_aktuell <= this->anzahlElemente_max) );
};
```

Instanziierung mit Hilfe des Kopierkonstruktors

```
class clsQueue schlangeCopy(schlange);
```

- In den runden Klammern wird die Quelle übergeben.
- Die Attribut-Werte des Quellobjekts schlange werden den Attributen des Objekts schlangeCopy zugewiesen.
- Das Quell- sowohl das Zielobjekt sind vom gleichen Typ „Klasse“.

Zeiger als Objektvariablen nutzen

```
class clsQueue *ptrQueue;

ptrQueue = &schlange;
ptrQueue->pop();

cout << "\nZeiger auf Schlange: ";
ptrQueue->writeSchlange();

delete ptrQueue;
```

Beispiele/cppOOP_004_clsQueue...

Zeiger vom Typ „Klasse“

```
class clsQueue *ptrQueue;
```

- Zeiger vom Typ einer Klasse.
- Zeiger werden immer durch das Sternchen gekennzeichnet.
- Der Zeiger zeigt auf den Anfang eines Speicherbereichs, in dem die Attribute des Objekts abgelegt sind.

Deklaration

	int	*ptrKopf	;
class	clsQueue	*ptrQueue	;
Datentyp		*name	;

- Das Sternchen kennzeichnet einen Zeiger.
- Um Fehler zu vermeiden, sollte das Sternchen immer direkt vor dem Namen des Zeigers geschrieben werden.

Initialisierung

ptrQueue	=	&schlange	;
zeiger	=	&variable	;

- Zeiger müssen mit einer Speicheradresse initialisiert werden.
- Der Zeiger und das Objekt sind von der gleichen Klasse.
- Der Adressoperator & gibt die Adresse einer Variablen zurück. In diesem Beispiel wird die Adresse auf ein Objekt an den Zeiger übergeben.

Aufruf von Methoden über eine Objektvariable

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

- Direkter Zugriff auf eine Methode.
- Mit Hilfe des Punktoperators verweist die Objektvariable direkt auf eine Methode.

Aufruf von Methoden über einen Zeiger

ptrQueue	->	push	(	1	)	;
zeiger	->	methodenname	(	parameterliste	)	;

- Indirekter Zugriff auf eine Methode.
- Mit Hilfe des Operators -> greift ein Zeiger auf eine Methode zu.
- Der Zeiger verweist auf ein Objekt vom Typ „Klasse“. Die aufgerufene Methode ist in dieser Klasse deklariert.

Objekte dynamisch erzeugen

```
class clsQueue *ptrQueue;

ptrQueue = new clsQueue(4);

ptrQueue->push(1);
ptrQueue->push(2);

cout << "\nZeiger auf eine neue Schlange: ";
ptrQueue->writeSchlange();

delete ptrQueue;
```

Beispiele/cppOOP_004_clsQueue...

Arbeiten mit new

```
class clsQueue *ptrQueue;

ptrQueue = new clsQueue(4);
```

- Mit Hilfe des Schlüsselwortes new wird Speicher angefordert. Der Speicher wird im Heap des Rechners bereitgestellt.
- Dem Schlüsselwort folgt die benötigte Größe des Speicherbereichs. In diesem Beispiel wird Speicher für ein Objekt der Klasse clsQueue angefordert.
- In runden Klammern folgt die Parameterliste. In Abhängigkeit der Parameterliste wird der entsprechende Konstruktor aufgerufen.

Objekt löschen

```
delete ptrQueue;
```

- Objekte, die mit Hilfe von new erzeugt wurden, müssen mit delete gelöscht werden.
- Das Schlüsselwort delete ruft den Destruktor einer Klasse auf.

Array von Objekten dynamisch erzeugen

```
class clsQueue *ptrVieleSchlangen;

ptrVieleSchlangen = new clsQueue[3];
ptrVieleSchlangen[0].push(1);
ptrVieleSchlangen[1].push(11);

cout << "\n2. Schlange von 3 Schlangen: ";
ptrVieleSchlangen[1].writeSchlange();

delete[] ptrVieleSchlangen;
```

Beispiele/cppOOP_004_clsQueue...

Arbeiten mit new

```
class clsQueue *ptrVieleSchlangen;

ptrVieleSchlangen = new clsQueue[3];
```

- Mit Hilfe des Schlüsselwortes new wird Speicher angefordert. Der Speicher wird im Heap des Rechners bereitgestellt.
- Dem Schlüsselwort folgt die benötigte Größe des Speicherbereichs. In diesem Beispiel wird Speicher für x Objekte der Klasse clsQueue angefordert.
- In den eckigen Klammern folgt die Anzahl der Objekte. In diesem Beispiel wird ein Array von drei Elementen erzeugt.

Instanziierung der Elemente

```
class clsQueue *ptrVieleSchlangen;

ptrVieleSchlangen = new clsQueue[3];
```

- Jedes Element enthält eine Instanz vom Typ „Klasse“.
- Die Instanz wird mit Hilfe des Standard-Konstruktors erzeugt.

Elemente in einem Array

ptrViele	[	3	]
zeiger	[	index	]

- Dem Namen des Zeigers 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 Element

ptrViele	[	3	]	.	push	(	1	)	;
zeiger	[	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?

Objekt löschen

```
delete[] ptrQueue;
```

- Objekte, die mit Hilfe von `new` erzeugt wurden, müssen mit `delete` gelöscht werden.
- Dem Schlüsselwort folgen die leeren eckigen Klammern als Kennzeichnung für die Löschung eines Arrays.
- Falls die Klammern vergessen werden, wird irgendetwas gelöscht.

Zeiger-Arithmetrik

```
class clsQueue *ptrVieleSchlangen;
ptrVieleSchlangen = new clsQueue[3];

class clsQueue *tmpPtrSchlange;
tmpPtrSchlange = ptrVieleSchlangen;
(tmpPtrSchlange + 2)->push(21);
(tmpPtrSchlange + 2)->push(22);
cout << "\n3. Schlange von 3 Schlangen: ";
(tmpPtrSchlange + 2)->writeSchlange();

delete tmpPtrSchlange;
delete[] ptrVieleSchlangen;
```

Beispiele/cppOOP_004_clsQueue...

Nutzung von Zeiger-Arithmetrik

```
(tmpPtrSchlange + 2)->push(21);
```

- Der Zeiger wird um (x * sizeof(Datentyp)) verschoben.
- Die neue Adresse wird mit Hilfe der Speichergröße des angegebenen Datentyps berechnet.
- Mit Hilfe des Elementverweises wird der Zeiger mit einer Methode verbunden.

Hinweis

```
class clsQueue *ptrVieleSchlangen;
ptrVieleSchlangen = new clsQueue[3];

class clsQueue *tmpPtrSchlange;
tmpPtrSchlange = ptrVieleSchlangen;
```

- Der Zeiger auf den Anfang des dynamisch erzeugten Speichers wird nicht verändert.
- Mit Hilfe dieses Zeigers muss der Speicherbereich freigegeben werden.
- Das Array wird immer mit einem temporären Zeiger durchlaufen. Dieser Zeiger wird mit der Anfangsadresse des Arrays initialisiert.