Vorkurs C++ Programmierung

Klassen

Letzte Stunde

- Speicherverwaltung
  - automatische Speicherverwaltung auf dem Stack
  - dynamische Speicherverwaltung auf dem Heap
  - new/new[] und delete/delete[]
- Speicherklassen: **`auto, static, const`**
  - **`static`** und **`const`** können noch viel mehr
- Konstanten
  - **`const`** und **`#define`**

Heute: Klassen

- Klassen sind das zentrale Konzept der Objektorientierten Programmierung (OOP)
- Zur OOP gehört noch viel mehr...
  - Stichworte: Vererbung und Polymorphismus
  - Vollständige OOP erst in der Vorlesung EI-2
- Hier:

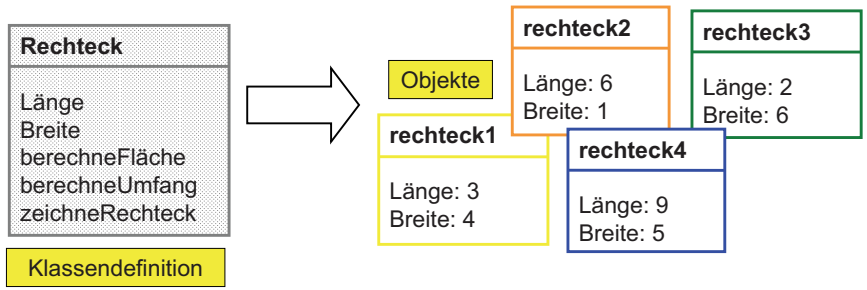
Klassen als Zusammenfassung logisch zusammengehöriger Variablen und Funktionen in einem eigenen Datentyp

Klassen

- Eine Klasse ist eine „Schablone“, mit welcher sich konkrete Instanzen („Objekte“) erzeugen lassen.
  - Analog: Einfacher Datentyp **`int`** → Variable **`int zahl = 17;`**
- Beispiel: Definition einer Klasse „Rechteck“
  → „Welche Eigenschaften hat ein Rechteck?“ (= Attribute)
  → „Was soll ein Rechteck „können“?“ (= Funktionen)
- Attribute: Länge und Breite
- Funktionen: Umfang berechnen, Flächeninhalt berechnen, gezeichnet werden

Klassen

- Beispiel: Klasse „Rechteck“:
  - Attribute: Länge und Breite
  - Funktionen: Fläche und Umfang berechnen, zeichnen



4 verschiedene Objekte (auch: „Instanzen“) der Klasse „Rechteck“ mit unterschiedlichen Attributwerten.

Vorschau...

Hier steht die Definition von „Rechteck“ drin!

„Konstruktor“, dem man Werte für die Länge und die Breite als Parameter übergeben kann

new liefert einen Pointer auf ein Objekt!

Aufruf der Funktion **berechneFlaeche()**, die wir für die Klasse definiert haben.

```
#include <iostream>
#include "Rechteck.h"

int main(int argc, char* argv[])
{
   Rechteck *rechteck1 = new Rechteck(3.0,4.0);
   float fFlaeche1 = rechteck1->berechneFlaeche();
   Rechteck *rechteck2 = new Rechteck(2.0,
   float fFlaeche2 = rechteck2->berechneFl
   if(fFlaeche1 < fFlaeche2) {
        std::cout << "Rechteck2 ist groesse
   }
   else if(fFlaeche1 > fFlaeche2) {
        std::cout << "Rechteck1 ist groesser";
   }
   else std::cout << "Die Rechtecke sind gleich gross";
   return 0;
}
```


Definition eigener Klassen

Wie definiert man eine Klasse?

- Üblich: Trennung von Deklarationen und Implementierung
- Header-Datei (Endung: .h) und Implementierung (.cpp)
- Deklarationen (Header-Datei):
  - Namen und Typen von Attributen (ohne Werte, da die Werte ja den Objekten zu Eigen sind)
  - Signaturen der Funktionen (ohne Rumpf!), des Konstruktors und Destruktors
- Implementierung (.cpp-Datei):
  - Konstruktor, Destruktor und Funktionen mit Rumpf
  - Hier steht drin, was wirklich passiert
- Vorteile der Trennung: Verbergen der Details, mehr Übersicht

Klassendeklaration: Header-Datei

```
//Datei: Rechteck.h

class Rechteck
{

public:
   Rechteck(float fL, float fB);

   ~Rechteck();

   float berechneFlaeche();
   float berechneUmfang();
   void zeichneRechteck();

private:
   float m_fLaenge;
   float m_fBreite;

};
```

Klassenname:
class *Klassenname* { ... };

Konstruktor:
Klassenname(); oder
Klassenname(Parameterliste);

Destruktor:
~Klassenname();

Funktions**signaturen**:
Returnwert Funktionsname(Parameterliste);

Attribute (Eigenschaften), auch „Membervariablen“:
Typ m_tAttributname;
(Konvention)

Semikolon!

Klassendeklaration: Header-Datei

```
//Datei: Rechteck.h

class Rechteck
{

public:
   Rechteck(float fL, float fB);

   ~Rechteck();

   float berechneFlaeche();
   float berechneUmfang();
   void zeichneRechteck();


private:
   float m_fLaenge;
   float m_fBreite;

};
```

public und **private** sind Schlüsselwörter, die die **Sichtbarkeit** von Attributen und Funktionen festlegen.

Implementierung (in der .cpp-Datei)

```
//Datei: Rechteck.h

class Rechteck
{

public:
   Rechteck(float fL, float fB);
   ~Rechteck();

   float berechneFlaeche();
   float berechneUmfang();
   void zeichneRechteck();

private:
   float m_fLaenge;
   float m_fBreite;
};
```

```
//Datei Rechteck.cpp
#include "Rechteck.h"
//Konstruktorimplementierung:
Rechteck::Rechteck(float fL, float fB)
{
   m_fLaenge = fL;
   m_fBreite = fB;
}
//Destruktorimplementierung
Rechteck::~Rechteck()
{
   //Speicher
   //freigeben
}
```

Bindet die Klassendefinition ein

Implementierung (in der .cpp-Datei)

```
//Datei: Rechteck.h

class Rechteck
{

public:
   Rechteck(float fL, float fB);
   ~Rechteck();

   float berechneFlaeche();
   float berechneUmfang();
   void zeichneRechteck();

private:
   float m_fLaenge;
   float m_fBreite;
};
```

```
//Datei Rechteck.cpp
#include "Rechteck.h"
//Konstruktorimplementierung:
Rechteck::Rechteck(float fL, float fB)
{
   m_fLaenge = fL;
   m_fBreite = fB;
}
//Destruktorimplementierung
Rechteck::~Rechteck()
{
   //Speicher
   //freigeben
}
```

Konstruktor: Initialisierung des zu erzeugenden Objekts mit **new**

Destruktor: Löschen eines Objekts mit **delete**

Implementierung (in der .cpp-Datei)

```
//Datei: Rechteck.h

class Rechteck
{

public:
   Rechteck(float fL, float fB);
   ~Rechteck();

   float berechneFlaeche();
   float berechneUmfang();
   void zeichneRechteck();

private:
   float m_fLaenge;
   float m_fBreite;
};
```

```
//Datei Rechteck.cpp
#include "Rechteck.h"
//Konstruktorimplementierung:
Rechteck::Rechteck(float fL, float fB)
{
   m_fLaenge = fL;
   m_fBreite = fB;
}
//Destruktorimplementierung
Rechteck::~Rechteck()
{
   //Speicher freigeben
}
```

:: definiert (u.a.) die Zugehörigkeit einer Funktion zu einer Klasse

Konstruktor

- „Bauanleitung“ für Objekte der Klasse

```
//.cpp-Datei
Rechteck::Rechteck(float fL, float fB)
{
   m_fLaenge = fL;
   m_fBreite = fB;
}
```

- hat immer den Namen der Klasse
- hat keinen Rückgabewert
- wird mit **`new`** aufgerufen
- braucht keine Parameter („Standardkonstruktor“):

```
//.cpp-Datei
Rechteck::Rechteck() {  }
```

...würde reichen.

Destruktor

```
//cpp-Datei
Rechteck::~Rechteck()
{
   //Speicher freigeben
}
```

- Wird mit **`delete`** aufgerufen
- Wird in der Klasse Speicher alloziert, muss er spätestens hier wieder freigegeben werden.
  → wird das Objekt der Klasse gelöscht, wird auch der Speicher gelöscht, den es belegt hat.
  → keine memory leaks!

Implementierung (in der .cpp-Datei)

```
//Datei: Rechteck.h

class Rechteck
{

public:
   Rechteck(float fL, float fB);
   ~Rechteck();

   float berechneFlaeche();
   float berechneUmfang();
   void zeichneRechteck();

private:
   float m_fLaenge;
   float m_fBreite;
};
```

```
//Datei Rechteck.cpp
#include "Rechteck.h"
//Konstruktorimplementierung:
Rechteck::Rechteck(float fL, float fB) { ... }
//Destruktorimplementierung
Rechteck::~Rechteck() { ... }


//Funktionen
float Rechteck::berechneUmfang()
{
   return 2 * (m_fLaenge + m_fBreite);
}

float Rechteck::berechneFlaeche()
{
   return m_fLaenge * m_fBreite;
}
```


Erzeugung von Objekten (auf dem Heap)

```
#include <iostream>
#include "Rechteck.h"


int main(int argc, char* argv[])
{
   Rechteck *erstesRechteck = new Rechteck(3.0, 4.0);
   float fFlaeche = erstesRechteck->berechneFlaeche();

   std::cout << nFlaeche;

   delete erstesRechteck;
   erstesRechteck = NULL;

   return 0;
}
```

1. (Pointer auf) Objekt erzeugen:
   `new Konstruktor(params);`
2. Funktion mit -> aufrufen (nur bei Pointertyp)

Erzeugung von Objekten (auf dem Heap)

```
#include <iostream>
#include "Rechteck.h"


int main(int argc, char* argv[])
{
   Rechteck *erstesRechteck = new Rechteck(3.0, 4.0);
   float fFlaeche = erstesRechteck->berechneFlaeche();

   std::cout << nFlaeche;

   delete erstesRechteck;
   erstesRechteck = NULL;

   return 0;
}
```

Wenn das Objekt nicht mehr gebraucht wird, Pointer löschen! **delete** ruft den Destruktor der Klasse auf.

Erzeugung von Objekten (auf dem Stack)

```
#include <iostream>
#include "Rechteck.h"


int main(int argc, char* argv[])
{
  Rechteck erstesRechteck(4.0,2.0);
  float fFlaeche = erstesRechteck.berechneFlaeche();

  std::cout << fFlaeche;

  return 0;
}
```

Bei einem „echten“ Objekt und bei Referenzen wird mit dem Punktoperator (.) auf Funktionen und Attribute zugegriffen.

Objekte als Parameter übergeben

- Beispiel: Vergleich zweier Rechtecke
  Deklarieren einer Funktion
  `bool isBigger(Rechteck *r);` in Rechteck.h
- Funktion nimmt einen Pointer auf ein Rechteck entgegen
  → Call by Value/Reference: Kopieraufwand, Änderungen nach außen!
- Implementierung in Rechteck.cpp:

```
bool Rechteck::isBigger(Rechteck *r)
{
  if(this->getFlaeche() > r->getFlaeche())
    return true;    //gibt „1“ aus
  else
    return false;   //gibt „0“ aus
}
```

this liefert immer einen Pointer auf das **aufrufende** Objekt

Objekte als Parameter übergeben

```
bool Rechteck::isBigger(Rechteck *r)
{
  if(this->getFlaeche() > r->getFlaeche())
      return true;  //gibt „1“ aus
  else
      return false; //gibt „0“ aus
}
```

Aufruf der Funktion an anderer Stelle (z.B. main.cpp):

```
int main(int argc, char* argv[])
{
  Rechteck *r1 = new Rechteck(3.9, 2.6);
  Rechteck *r2 = new Rechteck(1.0, 2.0);

  bool b = r1->isBigger(r2);

  std::cout << b;
  return 0;
}
```

Hier wird nur ein Pointer übergeben! Es müssen keine Daten kopiert werden!

Objekte als Parameter übergeben

- Jetzt mit Referenz:
  Implementierung in Rechteck.cpp:

```
bool Rechteck::isBigger(Rechteck &r)
{
  if(this->getFlaeche() > r.getFlaeche())
    return true;    //gibt „1“ aus
  else
    return false;   //gibt „0“ aus
}
```


Übersicht Parameterübergabe...

	Eine Funktion verlangt...		
	`...einen Pointer: void function(T *param)`	`...eine Referenz: void function(T &param)`	`...ein Objekt: void function(T param)`
Geg.: Pointer *p	`function(p)`	`function(*p)` `*: Dereferenzierung Liefert den Inhalt.`	`function(*p)` `*: Dereferenzierung Liefert den Inhalt.`
Geg.: Re-ferenz &r	`function(&r)` `&: Adressoperator Liefert die Adresse.`	`function(r)`	`function(r)`
Geg.: Objekt o	`function(&o)` `&: Adressoperator Liefert die Adresse.`	`function(o)`	`function(o)`

Call-by-Reference: **`function`** ändert die übergebenen Daten.

Call-by-Value: **`function`** kriegt nur eine Kopie. Änderungen nur lokal.

Objekte als Rückgabewert von Funktionen

- Objekte können wie einfache Datentypen Rückgabewert von Funktionen sein
- Hier sollte man ebenfalls Pointer und Referenzen benutzen!

```
//Funktion liefert einen Pointer auf das größte Rechteck
Rechteck* getBiggestRectangle(Rechteck *r1, Rechteck *r2)
{
  if(r1->berechneFlaeche() > r2->berechneFlaeche())
      return r1;
  else return r2;
}
```

Aufruf durch:

```
Rechteck *pRechteck1 = new Rechteck(3.0, 4.0);
Rechteck *pRechteck2 = new Rechteck(5.0, 6.0);
std::cout <<
    getBiggestRectangle(pRechteck1, pRechteck2)->getFlaeche();
```

Pointer auf das größere von beiden Rechtecken

Zusammenfassung

- Klassen: Zentrales Konzept der OOP
- Zusammenfassung/Kapselung von Daten und Funktionen
- Schablone zur Erzeugung konkreter Instanzen (Objekte)
- Trennung von Deklaration (Header-Datei, .h) und Implementierung (.cpp-Datei)
  - Einbinden mit dem #include-Befehl
- Konstruktor, Destruktor
  - Standardkonstruktor
- Übergabe von Objekten (Referenzen/Pointern) als Parameter
- Objekte als Rückgabewerte von Funktionen