Fachhochschule Bingen

Programmieren

OOP 1 / Einfache C++ Klassen

Prof. Dr. Maximilian Mengel,
Professur Programmiermethodik,
Grundlagen der Informatik und Multimedia
Gebäude 1, Raum 212
Tel.: 06721-409 152
E-Mail: mengel@fh-bingen.de

Abstrakter Datentyp: Daten und Funktionen

- Durch eigene Datentypen (in C per struct) können auf die jeweilige Anwendung angepaßte Variablen benutzt werden.
- Um mit diesen Datentypen zu arbeiten, müssen auch Funktionen und Prozeduren zur Manipulation entworfen werden
- Die Kombination aus neuem Datentyp und zugehöriger Funktionen und Prozeduren wird als **abstrakter Datentyp** bezeichnet
- Abstrakte Datentypen sollten in Modulen abgelegt werden

Modulare Programmierung: Vorteile/Nachteile

- Vorteile
  - Aufteilung eines Programms in mehrere (unabhängige) Module
  - Kapselung von Datentyp und Operatoren
  - Verbergen der Implementierung
- Nachteil
  - Datentyp ist exportiert
    - Genau eine Datenrepräsentation
    - Direkter Zugriff auf die Daten ist möglich
  - Keine eigenen Operatoren möglich: +, *, …
- => Objekt orientierte Sprachen (z.B. C++)

Objektorientierter Ansatz: Klassen

- Vollständige Kapselung eines abstrakten Datentyp in einem neuen Programmkonstrukt: der **Klasse**
- Eine Klasse umfasst:
  - Eigenschaften (Daten)
    - Graphisch: Größe, Farbe,
    - Auto: Geschwindigkeit
  - Methoden (Prozeduren/Funktionen)
    - Operationen zur Manipulation der Daten
    - Graphisch: verschiebe Position, ändere Farbe
    - Auto: beschleunige, bremse

Klasse versus Datentyp

- Eine Klasse umfasst:
  - Alles was auch ein Datentypen der prozeduralen Programmiersprachen beinhaltet
  - Eine Klasse definiert neben den Daten auch die zugehörigen Prozeduren und Funktionen, in C++ **Methoden** genannt
  - Die Daten (in C++ auch **Eigenschaften** genannt) sowie die zugehörigen Methoden bilden eine Einheit
- So wie von einem Datentyp Variablen gebildet werden werden von einer Klasse **Objekte instanziert**

Objekte

- Ein Objekt ist die konkrete Ausprägung einer Klasse:
  - Wie eine Variable belegt ein Objekt Speicher
  - In dem Speicher stehen die für das jeweilige Objekt spezifische Daten
  - Jedes Objekt besitzt seinen eigenen Speicherbereich für seine eigenen Daten
    - Ein Objekt besitzt eine Adresse (Stichwort: Zeiger!)
    - Genauso wie zwei Variablen die dieselben Inhalte haben deshalb trotzdem noch verschiedene Variablen sind ist dies auch bei Objekten
  - Die Manipulation der Objektdaten geschieht über die in der Klasse definierten Methoden

Objekt orientierte Programmierung

- Programmieren von Klassen
  - Welche Eigenschaften (Daten) werden benötigt
  - Welche Methoden (Funktionen/Prozeduren) werden für die Manipulation dieser Daten gebraucht
- Programmierung einer Anwendung
  - Erzeugen der Objekte
  - Objekte werden aufgerufen und rufen einander auf

Klassen

- Interface einer Klasse
  - Definiert die Eigenschaften und Methoden der Klasse
- Implementierung der Klasse
  - Umsetzung des Interface
- Abstimmung der Zugriffsrechte ist möglich
  - Private Methoden und Eigenschaften
    - Sind innerhalb der Klasse benutzbar
    - Von Außen nicht zugänglich
  - Allgemeine Methoden und Eigenschaften
    - Für jeden zugänglich

Ein- und Ausgabe in C++

- Neben der von C bekannten Funktionen zur Ein und Ausgabe existiert in C++ auch eine Objektorientierte Ein- und Ausgabe
  - Klasse ostream // Klasse zur Ausgabe
  - Standard-Instanzen: cout, cerr
  - Operator << zum Schreiben
  - Klasse istream // Klasse zur Eingabe
  - Standard-Instanz: cin
  - Operator >> zum Lesen
- #include <iostream.h>

Ein- und Ausgabe in C++: Beispiel

```
#include <iostream.h>
main()
{
   double km, liter;
   cout << "getankte Liter ? \n";
   cin >> liter;
   cout <<"gefahrene Kilometer?\n";
   cin >> km;

   cout << "Benzinverbrauch: ";
   cout << 100*liter/km;
   cout << " Liter je 100 Kilometer.\n";
}
```


Erstes Beispiel: Komplexe Zahlen - Konzept -

- Interface der Klasse complex:
  - Private Eigenschaften
    - Variable Realteil und Variable Imaginärteil (z.B. beide vom Typ double)
    - Oder alternativ /ergänzend Variable Betrag und Winkel (z.B. ebenfalls beide vom Typ double)
  - Allgemeine Methoden
    - getRealTeil()
    - getImagTeil()
    - setRealTeil()
    - setImagTeil()

C++ Klasse

- C++ Klasse ist so ähnlich wie ein struct aufgebaut
- Eine C++ Klasse
  - Beginnt mit dem Schlüsselwort **`class`**
  - Gefolgt von dem Klassennamen
  - Einer öffnenden geschweiften Klammer {
  - Den Eigenschaften (Datentyp & Eigenschaftenname) und dem Methoden
  - folgende Schlüsselworte steuern hierbei den Zugriff
    - public:
    - private:
    - protected:
  - Einer schließenden geschweiften Klammer } mit abschließendem Semikolon ;

Verbergen / Freigabe innerhalb einer Klassenhierachie

- Bedeutung der Schlüsselworte
  - public:
    - Von außen kann uneingeschränkt zugegriffen werden
  - privat:
    - Nur innerhalb der Klasse kann zugegriffen werden
  - protected
    - Innerhalb der Klasse selbst, sowie in allen abgeleiteten Klassen kann zugegriffen werden (Stichwort Vererbung; siehe später)

Eigenschaften & Methoden

- Die Eigenschaften werden analog zu struct-Datenfeldern aufgeschrieben
  - 1. Datentyp
  - 2. Name
  - 3. Semikolon
- Methoden-Implementierung kann analog zu „Funktionen/Prozeduren“ geschehen
  - 1. Rückgabewert
  - 2. Name
  - 3. Parameter: Call by Reference // Call by Value
  - 4. Implementierungsblock

Beispiel: Zaehler-Klasse

- Klasse Zähler:
  - Private Eigenschaften
    - int wert
  - Benutzbare Methoden
    - increment(), decrement(), init(), getwert()

```
class zaehler {
   private:
       int wert;
   public:
       void increment() {++wert;}
       void decrement() {--wert;}
       void init() {wert = 0;}
       int getwert(){return wert;}
   };
```


Implementierung von Methoden

- Die Implementierung von Methoden kann auf zwei Arten erfolgen
  - A) Direkt in der Klasse
  - B) Außerhalb der Klasse
    - In der Klasse wird nur der Prototyp definiert
    - Die Implementierung erfolgt separat, außerhalb der Klasse:
    - Bei der Implementierung wird der Klassenname mit dem Operator :: vor dem Methodennamen angegeben:

```
double complex::getReal()
{
      return re;
}
```

Beispiel: C++

```
class complex
    {
        double re, im;   // implizit private:
    public:
          void init(double neuerRe, double neuerIm);
          double getReal();
          double getImag();
          void setReal(double neuerRe) {re = neuerRe;}
          void setImag(double neuerIm) {im = neuerIm;}
    };

void complex::init(double neuerRe,double neuerIm)
    {
        re = neuerRe;
        im = neuerIm;
    }
double complex::getReal()      { return re; }
double complex::getImag()      { return im;}
```


Instanzierung und Aufruf von Methoden

- Um ein Objekt zu instanzieren kann man dieses analog zu einer normalen Variablen vereinbaren
- Methode und Objekt bilden eine Einheit
  - => setReal(3.5) allein kann nicht aufgerufen werden
- Beim Aufruf wird die Methode des entsprechenden Objekts mittels des Operators . aufgerufen

```
main()
{
   complex z1;
   z1.init(3.5, 6);
}
```

- Hierbei müssen die Zugriffsrechte beachtet werden!

Methoden: Konstruktor / Destruktor

- Konstruktoren und Destruktoren sind spezielle Methoden einer Klasse
  - Der Konstruktor wird beim Instanzieren des Objektes ausgeführt
  - Der Destruktor wird ausgeführt wenn ein Objekt aus dem Speicher entfernt werden soll.

```
if (...)
{
      complex z1;  // Konstruktor wird ausgeführt
      ...
      z1.setImag(2.4);
}                  // Destruktor wird ausgeführt
```


Namensgebung: Konstruktor / Destruktor

- Der Konstruktor einer Klasse heißt genauso wie die Klasse selbst
  - Im Konstruktor initialisiert man interne Variablen und fordert eventuell benötigten Speicherplatz an
- Der Destruktor heißt wie die Klasse selbst mit einem vorangestellten ~
  - Im Destruktor gibt man eventuell angeforderten Speicherplatz wieder frei
- Konstruktor und Destruktor besitzen keinen Rückgabewert (auch nicht void)

Beispiel Konstruktor/ Destruktor

```
class Feld {
   int elemente;
   int* f;
public:
   Feld(int anzahl);
   ~Feld();
   ...
}

Feld::Feld(int anzahl)
{
   elemente = anzahl;
   if (anzahl > 0)
       f = (int *) calloc(anzahl,sizeof(int));
}
Feld::~Feld()
{
   free(f);
}
```


New / Delete

- Um Objekte im Speicher zu erzeugen existiert die Operation **new**
  - new benutzt intern die *alloc-Funktionen
  - Im Gegensatz zu malloc() wird bereits der „richtige“ Datentyp zurückgegeben
  - Bei Mißerfolg wird NULL zurückgegeben
- Um Objekte zu löschen und deren Speicherplatz freizugeben existiert die Operation delete
- Um eine ganzes Array freizugeben nutzt man die Operation delete [ ]

Beispiel new / delete

```
#include <iostream.h>
#include <string.h>

main()
{
   char* c = new char[100];
   int* i = new int;

   strcpy(c,“Ein Text“);
   *i = 42;
   cout << c << “\n“;
   cout << *i << “\n“;

   delete i;
   delete [] c;
}
```