HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN MÜNCHEN

Kapitel 2

Objektorientierte Programmierung

Standardbibliothek: Strings, Container, Streams

Definition von neuen Klassen

Konstruktoren, Destruktoren

Vererbung

Einleitung

Gelegentlich merkt man, dass die Programmiersprache C etwas „in die Jahre gekommen ist“. Gewisse Dinge lassen sich nur umständlich programmieren.

Häufig kommt es auch vor, dass vermeintlich einfache Funktionen oder Programme unentdeckte Fehler enthalten, die dann mit etwas Pech durch Schadsoftware ausgenutzt werden. Etwa, um das jeweilige Programm zum Absturz zu bringen oder gar den betroffenen Rechner zu kompromittieren.

In drei Beispielen werden die Verarbeitung von Zeichenketten (Strings), die Arbeit mit Feldern und Vektoren sowie die Ein- und Ausgabe von Variablen in C gezeigt. Anschließend werden dieselben Beispiele auf eine andere (schönere?) Art in der Programmiersprache C++ gelöst.

Viele weitere Beispiele lassen sich leicht finden…

Einleitung, 1. Beispiel: Zeichenketten

```
/* (Schlechtes!!) Beispiel: Zeichenketten in C */
#include <stdio.h>
#include <string.h>

void append_str(char *str)
{
    strcat(str, " bla bla");
}

int main(void)
{
    int i;
    char str[100];
    printf("Zeichenkette eingeben: "); scanf("%s", str);
    printf("Integer-Zahl eingeben: "); scanf("%d", &i);

    append_str(str);
    printf("Zeichenkette: %s\nInteger-Zahl: %d\n\n", str, i);
    return 0;
}
```

1. *Was macht dieses C-Programm?*
2. *Welche Probleme haben sich in diesem C-Programm versteckt?*

Einleitung, 2. Beispiel: Ein- und Ausgabe

Ist dies nicht furchtbar…?!

```
#include <stdio.h>
#include <complex.h>
#include "matrix.h"

int main(void)
{
    int i; double d, re, im; float f; double _Complex dc; double m[2][2];

    printf("int:     "); scanf("%d" , &i);
    printf("float:   "); scanf("%f" , &f);
    printf("double:  "); scanf("%lf", &d);
    printf("complex: "); scanf("%lf %lf", &re, &im); dc = re + I * im;
    printf("2x2-Matrix: ");
    scanf("%lf %lf %lf %lf", &m[0][0], &m[0][1], &m[1][0], &m[1][1]);

    printf("int:    %d\n", i);
    printf("float:  %f\n", f);
    printf("double: %f\n", d);
    printf("complex: %f + %fi\n", creal(dc), cimag(dc));
    printf("matrix:\n"); m_print(2, m);
    return 0;
}
```

Einleitung, 3. Beispiel: Vektoren

```
#include <stdio.h>
#include <stdlib.h>

void zufall(double *par)  // Alternative: double par[]
{
    int i;
    for(i = 0; i < 10; ++i)
        *(par + i) = 2.0 * rand() / RAND_MAX - 1.0;  // par[i] = ...
}

int main(void)
{
    int i;
    double a[10];

    zufall(&a[0]);  // Alternative: zufall(a);
    for(i = 0; i < 10; ++i)
        printf("%10.6f\n", a[i]);
    return 0;
}
```

1. *Kann die Funktion „zufall" auch Vektoren mit mehr/weniger als 10 Elementen verarbeiten?*
2. *Kann man Vektoren auch „by value" statt „by reference" übergeben?*

Die C++-Standardbibliothek

Kaum ein Programmierer nutzt zur Softwareentwicklung eine „nackte" Programmiersprache. Stattdessen werden für Standardaufgaben zunächst leistungsfähige Bibliotheken entwickelt, diese bilden die Grundlage für die weitere Programmierung. Die Beschreibung der C++-Standardbibliothek umfasst ca. 2/3 des gesamten ISO-C++-Standards.

Die C++-Standardbibliothek bietet – neben vielen anderen Dingen – Klassen zum Speichern von Daten (z. B. **std::vector**), Klassen zur Arbeit mit Zeichenketten (z. B. **std::string**) und Klassen zur Ein- und Ausgabe auf dem Bildschirm und in Dateiform (z. B. **std::cout**).

```
std::string s1;
```

...oder...

```
using namespace std;
string s2;
```

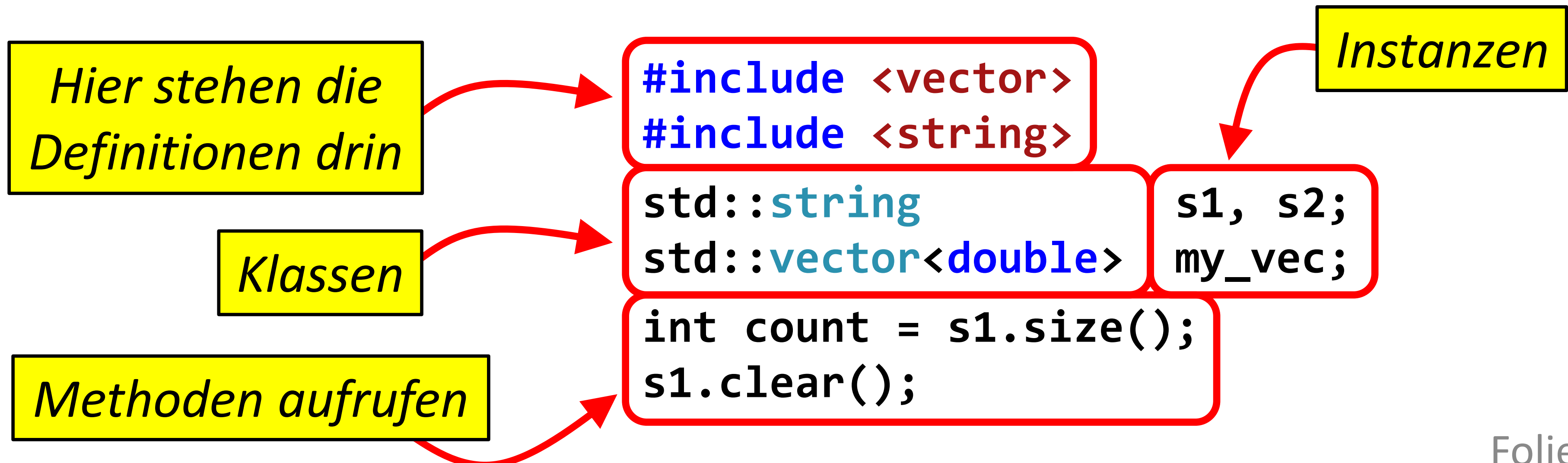
Die Standardklassen sind im **Namensraum std** definiert, um mögliche Namenskollisionen mit eigenen Definitionen zu verhindern.

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN MÜNCHEN

Objektorientierte Programmierung

Bei objektorientierten Sprachen können zusätzlich zu den eingebauten Variablentypen (int, float, char…) neue Typen (sog. **Klassen**, bspw. std::string) definiert werden. Nachdem eine Klasse implementiert wurde, können Variablen dieses Typs angelegt werden (**Instanzen**).

Klassen stellen Schnittstellen mit Funktionen (**Methoden**) und Daten (**Attribute**) zur Verfügung. Interne Daten und Abläufe – zum Beispiel die interne Speicherverwaltung – sind von außen nicht sichtbar.



Zeichenketten mit der C++-Standardbibliothek

Funktionsparameter und Rückgabewerte des Typs std::string sind kein Problem!

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
  string s1 = "eins", s2 = "zwei";  // String-Instanzen definieren
  string s3 = s1 + " und " + s2;    // Strings "addieren"
  cout << s3 << endl;               // Ausgabe von Strings

  getline(cin, s1);                 // Ganze Zeile einlesen
  cin >> s2;                        // Bis zum Leerzeichen einlesen

  int len = s3.size();              // Anzahl der Zeichen ermitteln
  for(int i = 0; i < len; ++i)      // Zeichen im String durchlaufen
  {
    char c = s3[i];                 // Einzelnes Zeichen ermitteln
    cout << c << endl;              // Einzelnes Zeichen ausgeben
  }
}
```

Ein- und Ausgabestreams mit der C++-Standardbibliothek

```
#include <iostream>     // Definitionen von cin, cout
#include <fstream>      // Definitionen von ofstream, ifstream
using namespace std;

int main()
{
    int i = -10;
    double d = 123.456;
    string s = " Hallo ", s1;
    cout << i << s << d << endl;              // Ausgabe auf Bildschirm

    ofstream my_file("c:/temp/test.txt");  // Ausgabedatei öffnen
    my_file << i << s << d << endl;           // Ausgabe in Datei

    cout << "Komplette Zeile eingeben:      ";  getline(cin, s);
    cout << "Zeichenkette ohne Leerzeichen: ";  cin >> s1;
    cout << "Bitte Integerzahl eingeben:    ";  cin >> i;
    cout << s << ", " << s1 << ", " << i << endl;
}
```

Containerklassen (Vektoren, Listen) mit der C++-Standardbibliothek

- *Es gibt etliche weitere Containerklassen, zum Beispiel std::list und std::set, siehe: http://www.cplusplus.com/reference/stl/*
- *Containerklassen als Funktionsparameter oder als Rückgabewert sind problemlos möglich.*

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;

int main()
{
    vector<int> iv;                         // Vektoren definieren,
    vector<string> sv;                      // Template-Schreibweise beachten

    iv.push_back(1234); iv.push_back(2345);  // Elemente ans Ende
    sv.push_back("aa"); sv.push_back("bb");  // des Vektors anhängen

    int ilen = iv.size();                   // Anzahl der Elemente ermitteln,
    for(int i = 0; i < ilen; ++i)           // alle Elemente im Vektor
        cout << iv[i] << endl;              // durchlaufen und ausgeben

    for(auto x : sv)                        // Alternative zum Durchlaufen
        cout << x << endl;                  // der Vektorelemente ab C++11
}
```

2.11. Standardbibliothek

Templates (engl.: Vorlagen, Schablonen)

Ein std::vector eignet sich zur Speicherung grundlegender Datentypen wie int, float oder double. Es können aber auch Zeichenketten des Typs std::string gespeichert werden. Oder sogar Instanzen von selbst definierten Klassen, die nicht Teil der C++-Standardbibliothek sind.

Bei der Definition von Vektor-Instanzen wird in eckigen Klammern angegeben, welcher Datentyp gespeichert werden soll:

```
using namespace std;
vector<double> v1;
v1.push_back(1.234);
v1.push_back(2.345);
```

```
using namespace std;
vector<string> v2;
v2.push_back("Hallo");
v2.push_back("Welt!");
```

Die Vektoren in der C++-Standardbibliothek sind sog. **Templates**. Vorteil: Es müssen nicht für alle möglichen Datentypen separate Vektoren programmiert werden (Integer-Vektor, String-Vektor…) sondern nur eine einzige allgemeine Vorlage.

Kapitel 2

Objektorientierte Programmierung

Standardbibliothek: Strings, Container, Streams

Definition von neuen Klassen

Konstruktoren, Destruktoren

Vererbung

Beispiel:

Definition einer Klasse für komplexe Zahlen mit Methoden zum Setzen und zum Abfragen von Real- und Imaginärteil. Später können weitere Methoden zum Setzen und Abfragen von Betrag und Winkel, für die vier Grundrechenarten sowie für die Ein- und Ausgabe über std::cin bzw. std::cout hinzugefügt werden.

(In der C++-Standardbibliothek steht übrigens bereits eine Klasse std::complex für komplexe Zahlen zur Verfügung. Dazu ist mittels #include <complex> die entsprechende Header-Datei einzubinden.)

Öffentliche Methoden:

- Zum Setzen von Real- und Imaginärteil,
- zum Abfragen von Real- und Imaginärteil.

Interne Attribute:

- Zwei double-Variablen für Real- und Imaginärteil

2.14. Definition von neuen Klassen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN MÜNCHEN

Aufgabe:

Fügen Sie weitere Methoden hinzu

- *zum Abfragen von Betrag und Winkel,*
- *zum Addieren und Subtrahieren,*
- *zur Ausgabe auf std::cout.*

```
#include <iostream>
using namespace std;

class Complex
{
private:
    double re, im;

public:
    void Set(double real, double imag) { re = real; im = imag; }
    double Real() { return re; }
    double Imag() { return im; }
};

int main()
{
    Complex c1, c2;
    c1.Set(1, 0);
    c2 = c1;
    cout << c1.Real() << ", " << c1.Imag() << endl;
    cout << c2.Real() << ", " << c2.Imag() << endl;
    return 0;
}
```

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN MÜNCHEN

Kapitel 2

Objektorientierte Programmierung

Standardbibliothek: Strings, Container, Streams

Definition von neuen Klassen

Konstruktoren, Destruktoren

Vererbung

Das **Anlegen neuer Instanzen** und das **Löschen nicht mehr benötigter Instanzen** sind „besondere Momente“ im Leben einer Klasse…!

```
int main()
{
    Complex comp1;
    cout << comp1.Real() << ", " << comp1.Imag() << endl;

    Complex comp2;
    comp2.Set(10.0, 20.0);
    .
    .
    .
```

Welche Werte werden hier eigentlich ausgegeben?

Kann man dies nicht auch kürzer schreiben?

Zusatzfrage:
Wie schafft es ein std::vector, dass der von ihm belegte Speicher freigegeben wird, wenn der Vektor nicht mehr benötigt wird?

```
class Complex
{
public:
    // Standard-Konstruktor
    Complex()
    {
        Set(0, 0);
    }

    // Zusätzlicher Konstruktor
    Complex(double real, double imag)
    {
        Set(real, imag);
    }

    // Destruktor
    ~Complex()
    {
        // cout << "Speicher wird wieder freigegeben" << endl;
    }

    ...
```

Beim Anlegen von neuen Instanzen sorgen Konstruktoren dafür, dass notwendige Initialisierungen durchgeführt und Attribute auf definierte Startwerte gesetzt werden.

Destruktoren geben zum Beispiel reservierten Speicherplatz wieder frei oder erledigen andere „Aufräumarbeiten“.

Kapitel 2

Objektorientierte Programmierung

Standardbibliothek: Strings, Container, Streams

Definition von neuen Klassen

Konstruktoren, Destruktoren

Vererbung

2.19. Vererbung


```
...

class Complex
{
    ...
};

ostream& operator<< (ostream& strm, Complex c)
{
    strm << "(" << c.Real() << ", " << c.Imag() << ")";
    return strm;
}

int main()
{
    Complex c1; c1.Set(1, 0);
    cout << c1 << endl;

    ofstream file("c:/temp/test.txt");
    file << c1 << endl;
    return 0;
}
```

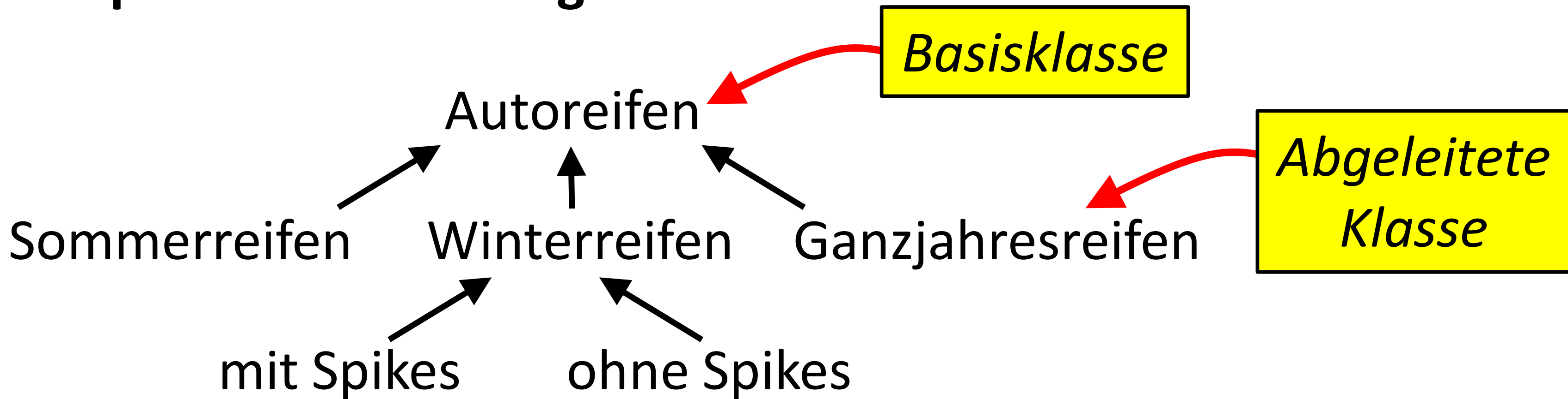
Wie kann es sein, dass der Ausgabe-Operator (<<) mit allen möglichen „Stream-Arten“ funktioniert (Ausgabe auf dem Bildschirm, in Dateien usw.)?

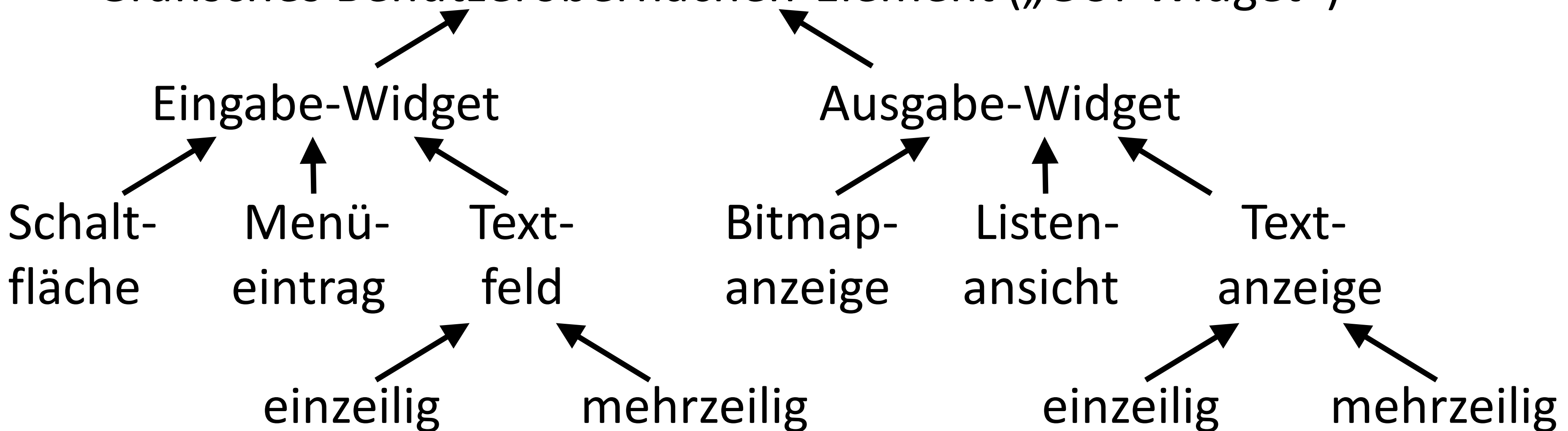
Antwort: Vererbung!

std::ostream

std::ofstream → *std::ostream* ← *std::ostringstream*

Beispiele für Vererbung:





2.21. Vererbung

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN MÜNCHEN

```
class Auto  // Basisklasse
{
private: double v;
public:  double Geschw() { return v; }
         void SetGeschw(double neu) { v = neu; }
         Auto() { SetGeschw(100); }
};

class E_Auto : public Auto  // Abgeleitete Klasse
{
private: double ladung;
public:  double Ladezustd() { return ladung; }
         void SetLadezustd(double neu) {ladung = neu; }
         E_Auto() { SetGeschw(50); SetLadezustd(100); }
};

void PrintGeschw(Auto a)  // Diese Funktion erwartet als Parameter ein Auto
{
    cout << "Geschwindigkeit: " << a.Geschw() << endl;
}

int main()
{
    Auto a;         E_Auto e;
    PrintGeschw(a); PrintGeschw(e);
}
```

Die Klasse E_Auto erbt die Attribute und Methoden von der Klasse Auto und fügt weitere Attribute und Methoden hinzu.

*Die Funktion PrintGeschw kann auch mit E_Auto-Instanzen aufgerufen werden, denn **E_Autos sind Autos!***