



Programmieren 3

C++

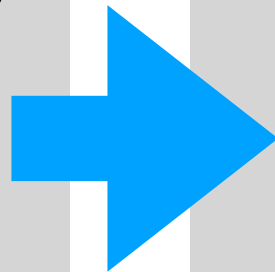
Vorlesung 02: Eingabe/Ausgabe, Container

Prof. Dr. Dirk Kutscher
Dr. Olaf Bergmann

Wiederholung

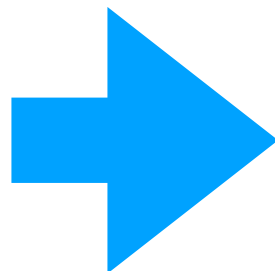
C: “Portabler Machine Code”

```
//add.c
int add(int i,int j)
{
    int p = i + j;
    return p;
}
```



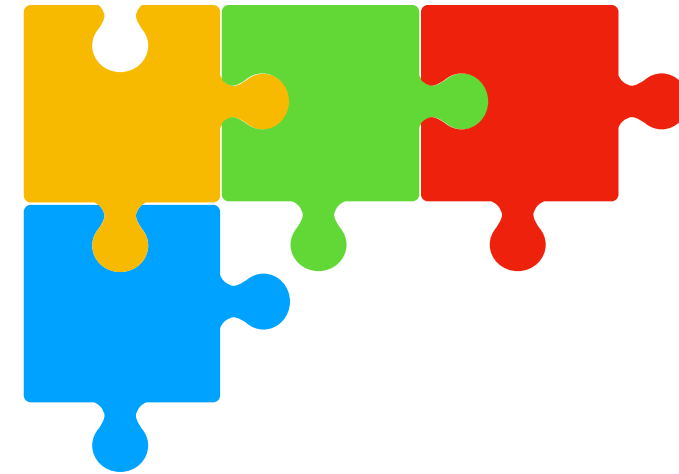
```
//add.s
.globl add
add:
    pushl %ebp
    movl %esp, %ebp
    subl $4, %esp    //create space for integer p
    movl 8(%ebp),%edx //8(%ebp) refers to i
    addl 12(%ebp), %edx //12(%ebp) refers to j
    movl %edx, -4(%ebp) //-4(%ebp) refers to p
    movl -4(%ebp), %eax //store return value in eax
    leave           //i.e. to movl %ebp, %esp; popl %ebp
    ret
```

add(10,20)



```
pushl $20
pushl $10
call add
```

Large-Scale Software-Entwicklung



- Modularisierung
 - Module wiederverwenden
 - Module erweitern
- Unabhängige Entwicklung
- Flexible Komposition

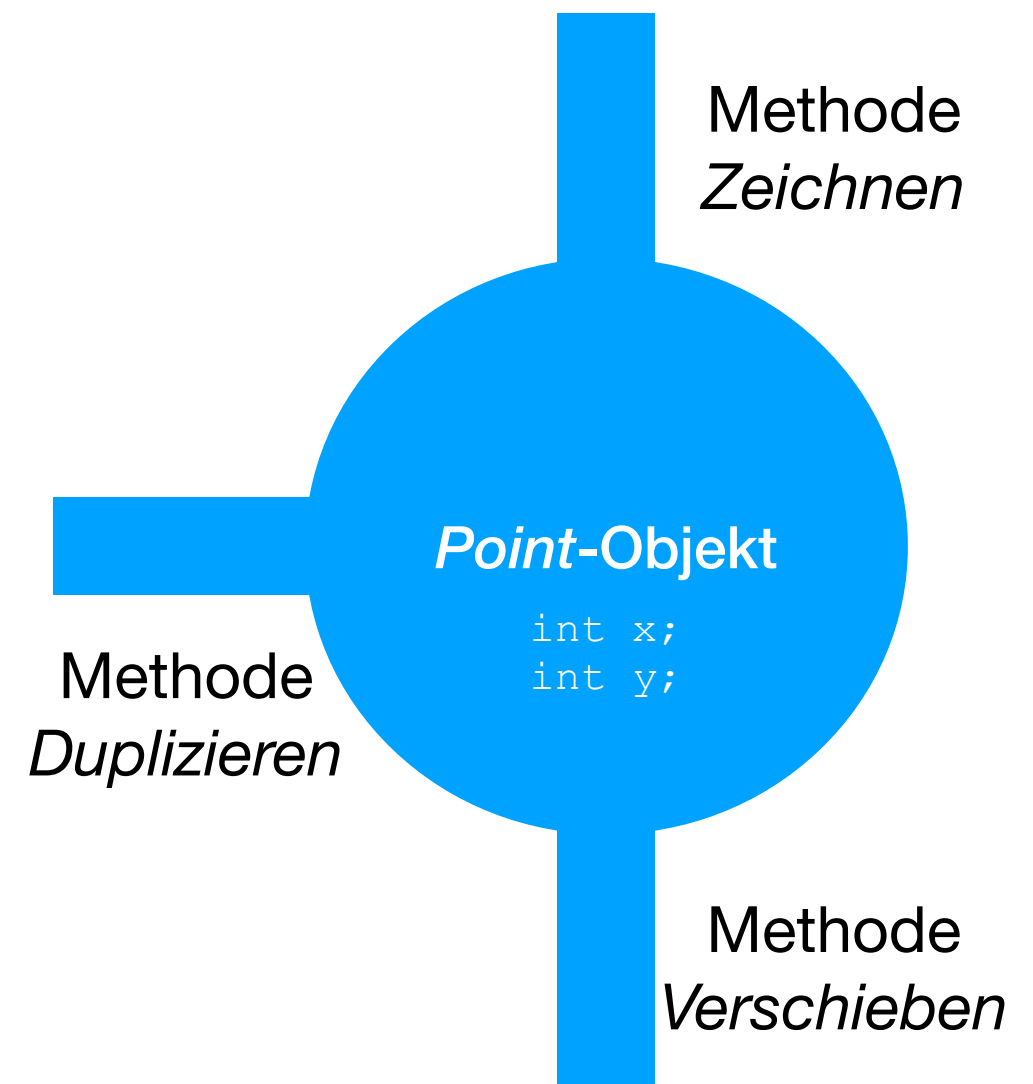
C: Wiederverwendung von Funktionen

```
//add.c
int add(int i,int j)
{
    int p = i + j;
    return p;
}
```

- Implementierung der Addition ist dem Nutzer egal (so lange Implementierung korrekt)
- Wichtig ist die Schnittstelle
 - Name der Funktion
 - Parameter
 - Rückgabewert

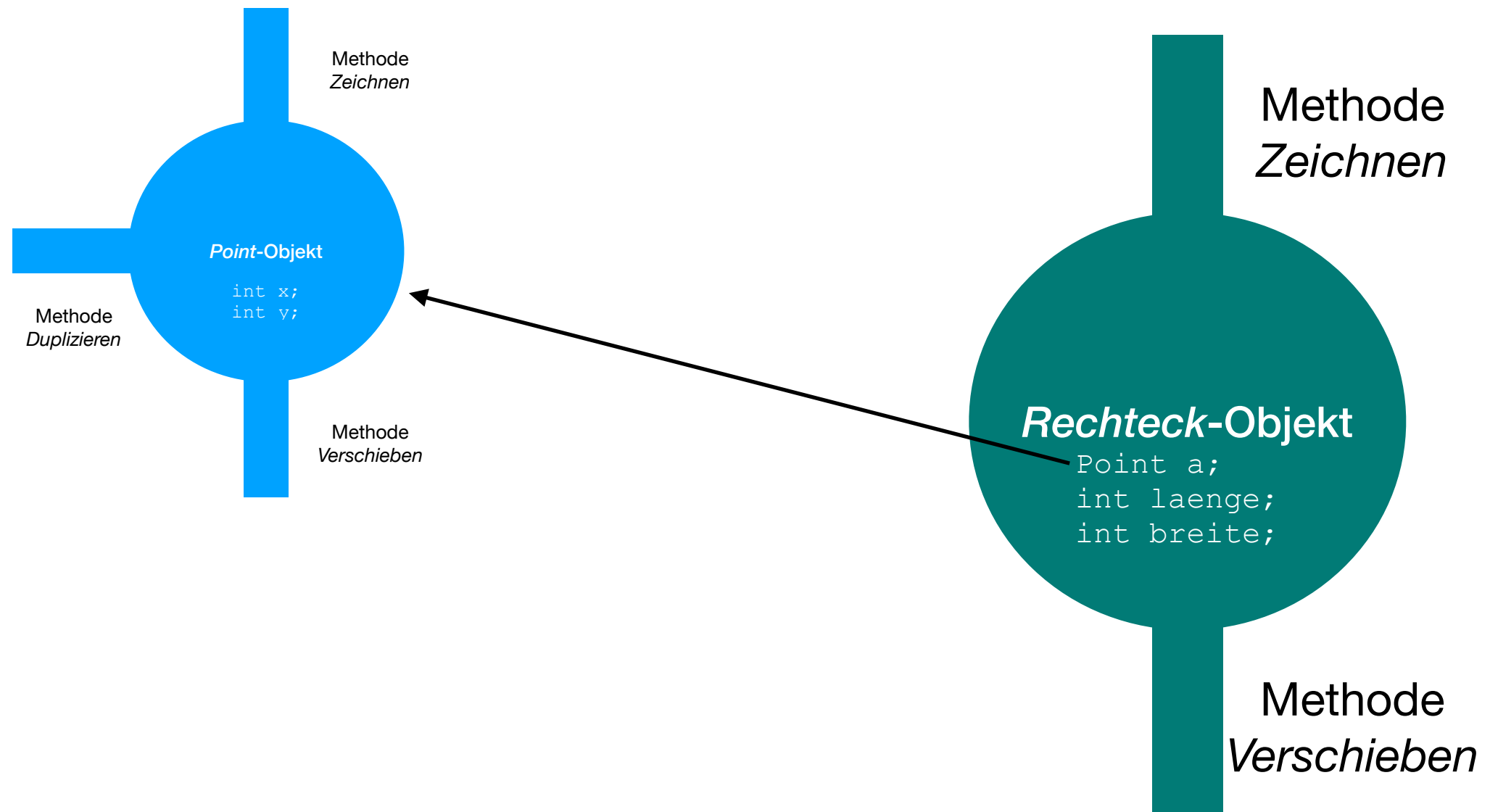
Objektorientierte Programmierung

- Vollwertige Abstrakte Datentypen
 - Die sich wie “eingebaute” Datentypen verhalten
- Interne Datenstruktur: ***Kapselung***
- Extern aufrufbare Funktionen (Methoden)



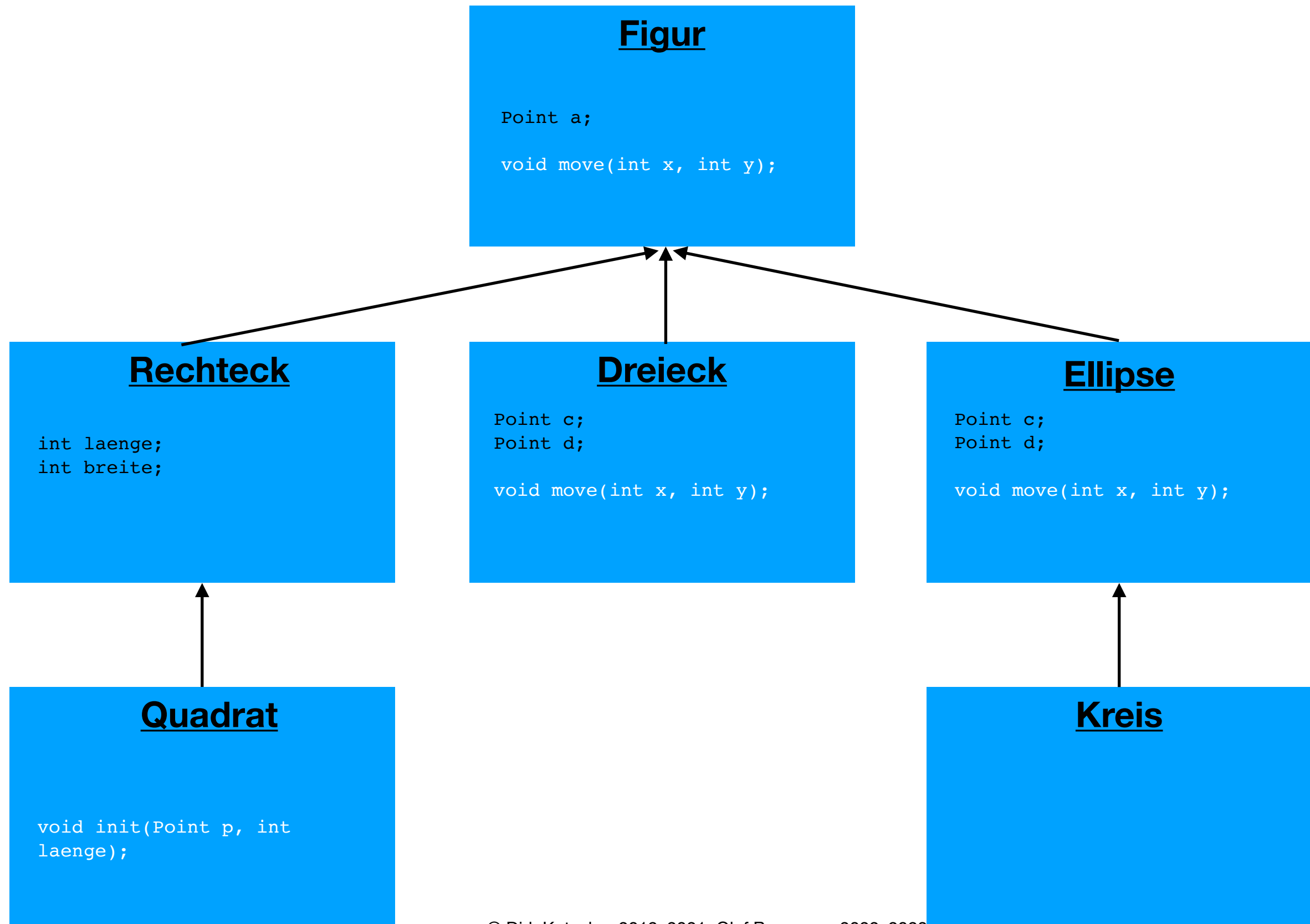
Objektorientierte Programmierung

Wiederverwendung durch *Komposition*



Objektorientierte Programmierung

Wiederverwendung durch *Vererbung*



Wiederverwendung durch Polymorphie

```
void moveAndDraw(Figure f, int x, int y) {  
    f.move(x,y);  
    f.draw();  
}
```

```
Quadrat q;  
Kreis k;  
  
moveAndDraw(q, 10, 5);  
moveAndDraw(k, 5, 1);
```

- Polymorphie: dasselbe Interface für eigentlich unterschiedliche Funktionen
 - Streng genommen, kein Kern-Feature von OOP
 - Wird aber bei C++ und anderen Sprachen so aufgefasst

Noch mehr Polymorphie: Operator-Overloading

- Problem: Bei größeren Projekten gehen dem Entwickler irgendwann die sinnvolle Funktionsnamen aus
- Namen wiederverwenden und anhand der Argumente entscheiden, welche Funktion aufgerufen wird (Overloading)
- U.a. können arithmetische, Bitwise-, Vergleichs-, logische, Inkrement/Dekrement-, Zuweisungs-Operatoren überladen werden

```
Vector2D Vector2D::operator+(const Vector2D& right)
{
    Vector2D result;
    result.set_x(x() + right.x());
    result.set_y(y() + right.y());
    return result;
}
```

Generisches Programmieren in C++ *Templates*

Ein generischer Listen-Typ

Elementtyp als Template-Parameter

- Templates: Funktionen und Klassen als “Schablonen” definieren
- Für beliebige Typen nach Bedarf instanziiieren

```
#include <algorithm>
#include <iostream>
#include <list>

int main()
{
    // Create a list containing integers
    std::list<int> l = { 7, 5, 16, 8 };

    // Add an integer to the front of the list
    l.push_front(25);
    // Add an integer to the back of the list
    l.push_back(13);

    // Insert an integer before 16 by searching
    auto it = std::find(l.begin(), l.end(), 16);
    if (it != l.end()) {
        l.insert(it, 42);
    }

    // Iterate and print values of the list
    for (int n : l) {
        std::cout << n << '\n';
    }
}
```

Namespaces

Library A

```
namespace A {  
    class Circle {  
        ...  
    }  
}
```

Library B

```
namespace B {  
    class Circle {  
        ...  
    }  
}
```

```
#include "liba.h"  
#include "libb.h"  
  
int main (int argc, char** argv) {  
    A::Circle kreis1;  
    B::Circle kreis2;  
    // ...  
    return 0;  
}
```

- Plus weitere Mechanismen (z.B. um Definitionen in Namespaces ohne Präfix zu verwenden)

Exceptions in C++

```
int dividiere(int dividend, int divisor) {  
    if(divisor==0)  
        throw(-1);  
    else  
        return (dividend/divisor);  
}  
  
int main(int argc, char** argv){  
    int result;  
    try {  
        result=dividiere(10/0);  
    }  
    catch (int) {cout << "oops";}   
    return 0;  
}
```

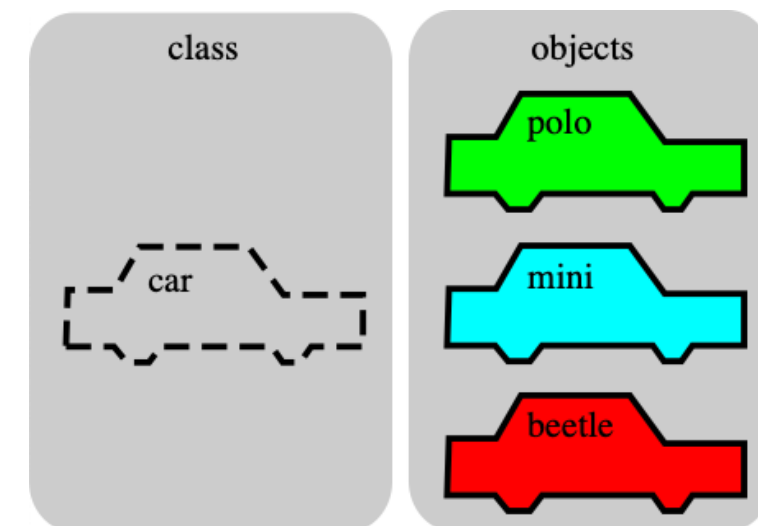
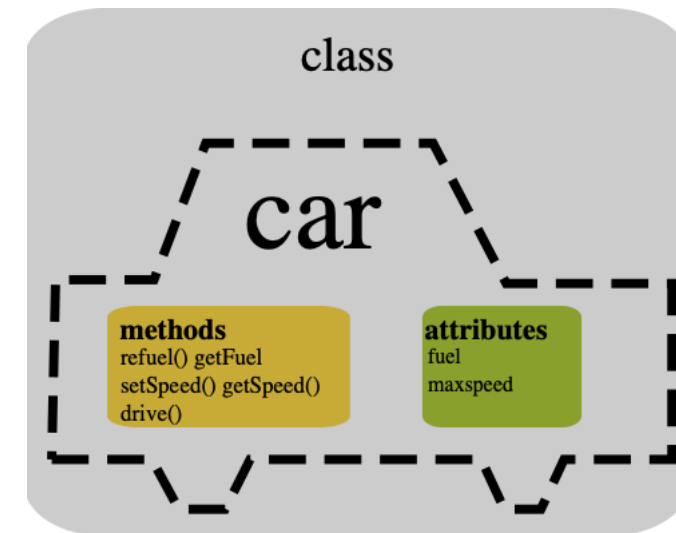
- Strukturierte Fehlerbehandlung für Laufzeitfehler
- Auch über mehrere Aufrufebenen hinweg
- Module, Libraries usw. können eigene Exception-Typen definieren (und dokumentieren)

Heute

- Basics, die man benötigt, um erste praktische Schritte zu machen
 - Standardbibliothek
 - IO
 - Container
 - Strings
 - Pointer vs. Referenzen

Objektorientierte Analyse und Design

- Die Welt als Klassen und Objekte modellieren
- Über Beziehungen zwischen Klassen und Objekten nachdenken
 - Was sind Eigenschaften / Member?
 - Was sind Vererbungsbeziehungen?
- Passt nicht immer, aber man kommt damit recht weit
- Braucht etwas Erfahrung



Definitionen vs. Deklarationen

Deklaration

```
extern int a;                                     // Declaring a variable a without defining it
struct tagExample { int a; int b; };             // Declaring a struct
int myFunc (int a, int b);                       // Declaring a function
```

```
int a;
int b = 0;
int myFunc (int a, int b) { return a + b; }
struct tagExample example;
```

Definition

```
// my first program in C++
#include <iostream>
int main()
{
    std::cout << "Hello World!";
}
```

Lädt die in iostream enthaltenen Deklarationen

Standardbibliothek

- Tools & Mechanismen zur Interaktion mit Betriebssystem
- Ein-/Ausgabe
- Strings
- Container (Listen, Vektoren usw.)
- ... und vieles mehr
- Wie bei C (`stdio.h` usw.) — nur viel umfangreicher

C++ Standard Library

- [Input/output](#)
- [Strings](#)

Standard Template Library

- [algorithm](#)
- [functional](#)
- [Sequence containers](#)
- [Associative containers](#)
- [Unordered associative containers](#)

C standard library

- [Data types](#)
- [Character classification](#)
- [Strings](#)
- [Mathematics](#)
- [File input/output](#)
- [Date/time](#)
- [Localization](#)
- [Memory allocation](#)
- [Process control](#)
- [Signals](#)
- [Alternative tokens](#)

Miscellaneous headers:

- [<assert.h>](#)
- [<errno.h>](#)
- [<setjmp.h>](#)
- [<stdarg.h>](#)

V•T•E

Hello World

```
// my first program in C++  
#include <iostream>  
  
int main() {  
    std::cout << "Hello World!";  
}
```

Hello World

Deklarationen aus der Datei `iostream` einbinden
Compiler kennt den Suchpfad für System-Header-Dateien

```
// my first program in C++  
#include <iostream>  
  
int main() {  
    std::cout << "Hello World!";  
}
```

Objekt für die Default-Ausgabe (z. B. Konsole)

Ausgabe-Operator

Eingabe

```
// i/o example

#include <iostream>
using namespace std;

int main ()
{
    int i;
    cout << "Please enter an integer value: ";
    cin >> i;
    cout << "The value you entered is " << i;
    cout << " and its double is " << i*2 << ".\n";
    return 0;
}
```

Eingabe

Wir verwenden alle Deklarationen aus dem Namespace std ohne Präfix

```
// i/o example

#include <iostream>
using namespace std;

int main ()
{
    int i;
    cout << "Please enter an integer value: ";
    cin >> i;
    cout << "The value you entered is " << i;
    cout << " and its double is " << i*2 << ".\n";
    return 0;
}
```

Objekt für die Default-Eingabe (z.B. Konsole)

Eingabe

Wir verwenden alle Deklarationen aus dem Namespace std ohne Präfix

```
// i/o example

#include <iostream>
using namespace std;

int main ()
{
    int i;
    cout << "Please enter an integer value: ";
    cin >> i;
    cout << "The value you entered is " << i;
    cout << " and its double is " << i*2 << ".\n";
    return 0;
}
```

Objekt für die Default-Eingabe (z.B. Konsole)


Eingabe-Operator

Dateien

```
#include <fstream>
int main()
{
    std::ofstream file("file.txt");
    file << "Hello, world!" << std::endl;
}
```

Dateien

Klasse für schreibbare Dateien




```
#include <fstream>
int main()
{
    std::ofstream file("file.txt");
    file << "Hello, world!" << std::endl;
}
```


Dateien

Klasse für schreibbare Dateien

Neues Objekt — lokale Variable

```
#include <fstream>
int main()
{
    std::ofstream file("file.txt");
    file << "Hello, world!" << std::endl;
}
```



Dateien

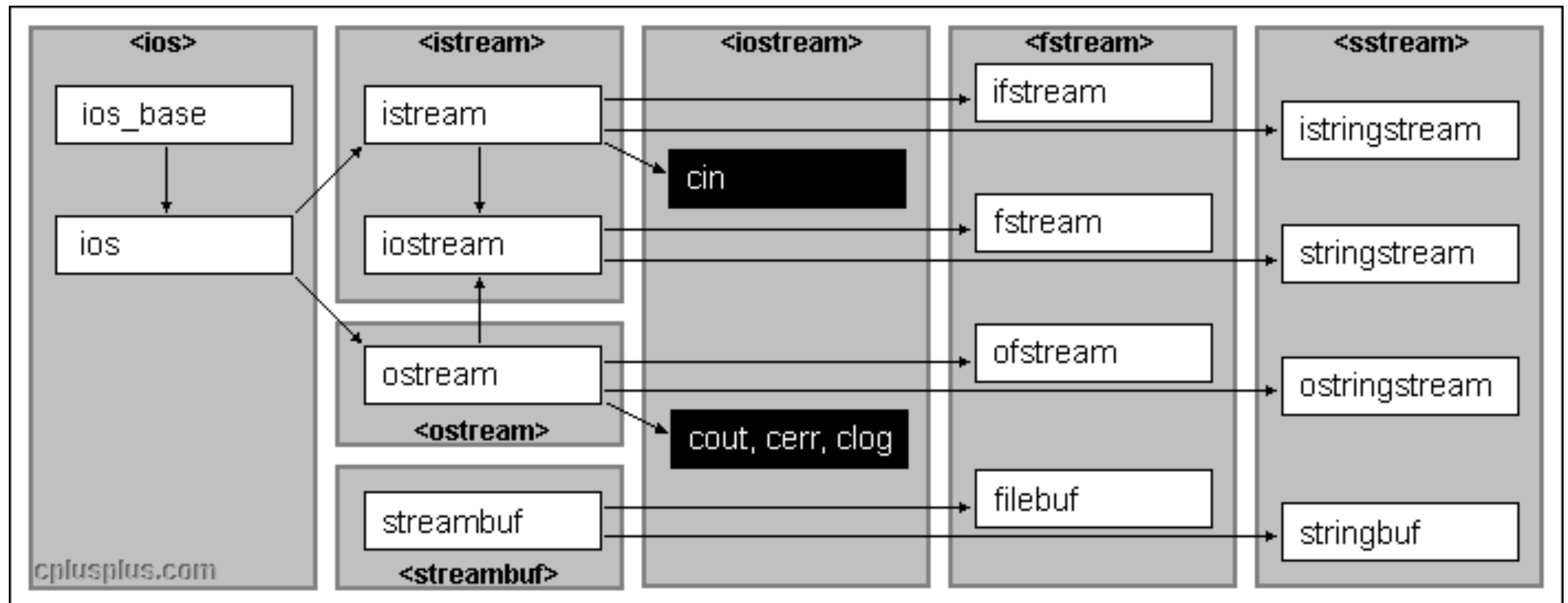
Klasse für schreibbare Dateien

Neues Objekt — lokale Variable

```
#include <fstream>
int main()
{
    std::ofstream file("file.txt");
    file << "Hello, world!" << std::endl;
}
```

Funktion, die portables
Zeilenende erzeugt

iostream-Klassen



Arrays in C

```
// Program to take 5 values from the user and store them in an array
// Print the elements stored in the array
#include <stdio.h>
int main() {
    int values[5];
    const size_t n_values = sizeof(values)/sizeof(values[0]);
    printf("Enter 5 integers: ");
    // taking input and storing it in an array
    for(int i = 0; i < n_values; ++i) {
        scanf("%d", &values[i]);
    }
    printf("Displaying integers: ");
    // printing elements of an array
    for(int i = 0; i < n_values; ++i) {
        printf("%d\n", values[i]);
    }
    return 0;
}
```

Arrays in C

```
// Program to take 5 values from the user and store them in an array
// Print the elements stored in the array
#include <stdio.h>
int main() {
    int values[5];
    const size_t n_values = sizeof(values)/sizeof(values[0]);
    printf("Enter 5 integers: ");
    // taking input and storing it in an array
    for(int i = 0; i < n_values; ++i) {
        scanf("%d", &values[i]);
    }
    printf("Displaying integers: ");
    // printing elements of an array
    for(int i = 0; i < n_values; ++i) {
        printf("%d\n", values[i]);
    }
    return 0;
}
```

Anzahl der Elemente
in values berechnen

Pre-increment operator:
i wird erst erhöht, und der
Ausdruck liefert den **neuen Wert**
von i

Vectors in C++

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
    vector<int> values(5);

    cout << "Enter 5 integers: ";
    // taking input and storing it in a vector
    for(int i = 0; i < values.size(); ++i) {
        cin >> values[i];
    }
    cout << "Displaying integers: ";
    // printing elements of an array
    for(int i = 0; i < values.size(); ++i) {
        cout << values[i] << endl;
    }
    return 0;
}
```

Vectors in C++

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
    vector<int> values(5);

    cout << "Enter 5 integers: ";
    // taking input and storing it in a vector
    for(int i = 0; i < values.size(); ++i) {
        cin >> values[i];
    }
    cout << "Displaying integers: ";
    // printing elements of an array
    for(int i = 0; i < values.size(); ++i) {
        cout << values[i] << endl;
    }
    return 0;
}
```

Container-Typ vector



Vectors in C++

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
    vector<int> values(5);

    cout << "Enter 5 integers: ";
    // taking input and storing it in a vector
    for(int i = 0; i < values.size(); ++i) {
        cin >> values[i];
    }
    cout << "Displaying integers: ";
    // printing elements of an array
    for(int i = 0; i < values.size(); ++i) {
        cout << values[i] << endl;
    }
    return 0;
}
```

Container-Typ vector



Element-Typ int



Vectors in C++

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
    vector<int> values(5);

    cout << "Enter 5 integers: ";
    // taking input and storing it in a vector
    for(int i = 0; i < values.size(); ++i) {
        cin >> values[i];
    }

    cout << "Displaying integers: ";
    // printing elements of an array
    for(int i = 0; i < values.size(); ++i) {
        cout << values[i] << endl;
    }

    return 0;
}
```

Container-Typ **vector**

Initiale Größe: 5 Elemente


Element-Typ **int**

Arrays kopieren

```
// Program to take 5 values from the user and store them in an array
// Print the elements stored in the array
#include <stdio.h>
int main() {
    int values[5];
    const size_t n_values = sizeof(values)/sizeof(values[0]);
    int copyOfValues[n_values];
    printf("Enter 5 integers: ");
    // taking input and storing it in an array
    for(int i = 0; i < n_values; ++i) {
        scanf("%d", &values[i]);
    }
    copyOfValues=values;
    printf("Displaying integers: ");
    // printing elements of an array
    for(int i = 0; i < n_values; ++i) {
        printf("%d\n", copyOfValues[i]);
    }
    return 0;
}
```

Arrays kopieren

```
// Program to take 5 values from the user and store them in an array
// Print the elements stored in the array
#include <stdio.h>
int main() {
    int values[5];
    const size_t n_values = sizeof(values)/sizeof(values[0]);
    int copyOfValues[n_values];
    printf("Enter 5 integers: ");
    // taking input and storing it in an array
    for(int i = 0; i < n_values; ++i) {
        scanf("%d", &values[i]);
    }
    copyOfValues=values;
    printf("Displaying integers: ");
    // printing elements of an array
    for(int i = 0; i < n_values; ++i) {
        printf("%d\n", copyOfValues[i]);
    }
    return 0;
}
```



Geht das?

Arrays kopieren: korrekte Fassung

```
// Program to take 5 values from the user and store them in an array
// Print the elements stored in the array
#include <stdio.h>
int main() {
    int values[5];
    const size_t n_values = sizeof(values)/sizeof(values[0]);
    int copyOfValues[n_values];
    printf("Enter 5 integers: ");
    // taking input and storing it in an array
    for(int i = 0; i < n_values; ++i) {
        scanf("%d", &values[i]);
    }

    for(int i = 0; i < n_values; ++i)
        copyOfValues[i]=values[i];

    printf("Displaying integers: ");
    // printing elements of an array
    for(int i = 0; i < n_values; ++i) {
        printf("%d\n", copyOfValues[i]);
    }
    return 0;
}
```

**Arrays in C müssen
elementweise kopiert werden**

Einen Vector kopieren in C++

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
    vector<int> values(5);
    vector<int> copyOfValues;

    cout << "Enter 5 integers: ";
    // taking input and storing it in a vector
    for(int i = 0; i < values.size(); ++i) {
        cin >> values[i];
    }

    copyOfValues=values;

    cout << "Displaying integers: ";
    // printing elements of an array
    for(int i = 0; i < copyOfValues.size(); ++i) {
        cout << copyOfValues[i] << endl;
    }
    return 0;
}
```