C++ Zusammenfassung

Inhalt

[1. Basics 4](#_Toc124530662)

[1.1 Ein-/Ausgabe 4](#_Toc124530663)

[1.2 Formatierte Ausgabe 5](#_Toc124530664)

[1.2.1 Beispiel Formatierte Ausgabe 5](#_Toc124530665)

[1.3 Referenzen 8](#_Toc124530666)

[1.3.1 Beispiel SWAP call by reference 8](#_Toc124530667)

[1.3.2 Beispiel Array (max, min, avg) 8](#_Toc124530668)

[1.3.3 Vergleich Zeiger/Referenzen 9](#_Toc124530669)

[1.4 Const Zeiger 9](#_Toc124530670)

[1.4.1 Zeiger auf const (Read-only-Zeiger) 9](#_Toc124530671)

[1.4.2 Konstante Zeiger 9](#_Toc124530672)

[1.4.3 Konstanter Read Only Zeiger 9](#_Toc124530673)

[1.4.4 Beispiel const Zeiger 10](#_Toc124530674)

[1.5 Const Referenz 10](#_Toc124530675)

[1.5.1 Fehler bei const pointer/referencen 10](#_Toc124530676)

[1.6 Dynamische Speicher 12](#_Toc124530677)

[1.6.1 Normal 12](#_Toc124530678)

[1.6.2 Array 12](#_Toc124530679)

[1.6.3 Beispiel dynamische Speicher 12](#_Toc124530680)

[1.7 Funktionszeiger 14](#_Toc124530681)

[1.7.1 const Funktionszeiger 14](#_Toc124530682)

[1.8 Default Parameter 14](#_Toc124530683)

[1.9 Inline Funktion 15](#_Toc124530684)

[1.10 Typplatzhalter auto 15](#_Toc124530685)

[1.11 Bereichbasierte for schleife 16](#_Toc124530686)

[2. Objektorientiertes Programmieren 17](#_Toc124530687)

[2.1 Assosation; Agregation 17](#_Toc124530688)

[2.2 Definition von Klassen/Strukt 18](#_Toc124530689)

[2.3 Memberfunktionen (Implementierung in CPP-Datei) 18](#_Toc124530690)

[2.3.1 const Memberfunktion 18](#_Toc124530691)

[2.3.2 Beispiel Memberfunktion Datum 19](#_Toc124530692)

[2.4 inline Methoden 20](#_Toc124530693)

[2.5 Instanzen (Speicher statisch, dynamisch) 20](#_Toc124530694)

[2.6 Konstruktor: Initialisierungsliste + Default 20](#_Toc124530695)

[3. Überladung von Operatoren 21](#_Toc124530696)

[3.1 Ausgabe-Operator 22](#_Toc124530697)

[3.2 [] Operator 22](#_Toc124530698)

[3.3 Weitere Operatoren 23](#_Toc124530699)

[3.4 Präfix/Postfix 23](#_Toc124530700)

[**3.5** The Big 3 24](#_Toc124530701)

[4. Vererbung 25](#_Toc124530702)

[5. Polymorphie 26](#_Toc124530703)

[5.1 Cast Operatoren 26](#_Toc124530704)

[5.1.1 Static Cast 26](#_Toc124530705)

[5.1.2 Dynamic Cast 26](#_Toc124530706)

[5.1.3 Const Cast 27](#_Toc124530707)

[5.1.4 Reinterpret Cast 27](#_Toc124530708)

[5.1.5 Beispiel Cast 28](#_Toc124530709)

[5.2 Typeid 29](#_Toc124530710)

[5.3 virtual Methode 29](#_Toc124530711)

[5.3.1 Abstrakte virtual Klassen 29](#_Toc124530712)

[6. Exceptions 30](#_Toc124530713)

[6.1 Try-cath-Block 30](#_Toc124530714)

[7. Templates 32](#_Toc124530715)

[7.1 Klassentemplates 33](#_Toc124530716)

[8. Beispiele 34](#_Toc124530717)

[8.1 Pascal Dreieck 34](#_Toc124530718)

[8.2 Funktionszeiger Trapezregel 35](#_Toc124530719)

[8.3 Funktionszeiger Complex struct 35](#_Toc124530720)

[8.4 Default Konstruktor 36](#_Toc124530721)

[8.5 Klasse Uhr 37](#_Toc124530722)

[8.6 Einfach Verkettete Liste 38](#_Toc124530723)

[8.7 Operatoren + Rule of Three 41](#_Toc124530724)

[8.8 Klasse: Vater Figur, Kind Kreis, Kind Rechteck 43](#_Toc124530725)

[8.9 Klasse ILebewesen Schnittstelle 44](#_Toc124530726)

[8.10 Klasse Fahrzeug (zu schwer) 46](#_Toc124530727)

[8.11 IZweipol (schaltung RLC) 50](#_Toc124530728)

[8.12 Observer 52](#_Toc124530729)

[8.13 Bubble Sort 54](#_Toc124530730)

[8.14 Template Stack 54](#_Toc124530731)

[9. Praktikum 58](#_Toc124530732)

[9.1 P1 58](#_Toc124530733)

[9.1.1 Komponente 58](#_Toc124530734)

[9.1.2 Bohrung 58](#_Toc124530735)

[9.1.3 Fraesung 59](#_Toc124530736)

[9.1.4 KomponentenElement 59](#_Toc124530737)

[9.1.5 KomponentenListe 60](#_Toc124530738)

[9.1.6 Main 62](#_Toc124530739)

[9.2 P2 63](#_Toc124530740)

[9.2.1 IKomponenten 63](#_Toc124530741)

[9.2.2 IKomponentenElement 64](#_Toc124530742)

[9.2.3 Komponente 64](#_Toc124530743)

[9.2.4 Bohrung 65](#_Toc124530744)

[9.2.5 Fraesung 66](#_Toc124530745)

[9.2.6 Iterator 67](#_Toc124530746)

[9.2.7 DeList 67](#_Toc124530747)

[9.2.8 Werkstueck 70](#_Toc124530748)

[9.2.9 Main 72](#_Toc124530749)

[9.3 P3 74](#_Toc124530750)

[9.3.1 TElement 74](#_Toc124530751)

[9.3.2 Iterator 74](#_Toc124530752)

[9.3.3 DeList 74](#_Toc124530753)

[9.3.4 Werkstueck 77](#_Toc124530754)

[9.3.5 Serializer 80](#_Toc124530755)

[9.3.6 Komponenten Änderungen 81](#_Toc124530756)

[9.3.7 Bohrung Änderungen 81](#_Toc124530757)

[9.3.8 Fraesung Änderungen 81](#_Toc124530758)

[9.3.9 Main Änderungen 81](#_Toc124530759)

# Basics

## Ein-/Ausgabe

|  |  |  |
| --- | --- | --- |
| #include | <iostream> | |
| *cin* | Standard-Eingabestrom | cin >> Variable |
| *cout* | Standard-Ausgabestrom | cout << Ausdruck |
| *cerr* | Strom fur ¨ ungepufferte Fehlerausgabe |  |
| *clog* | Strom fur ¨ gepufferte Fehlerausgabe |  |

|  |  |
| --- | --- |
| cin |  |
| cin >> var1 >> var2 ... >> varn; |  |
| cin.ignore(n, c); | maximal n Eingabezeichen bis zum Zeichen c ignorieren |
| while (cin >> wert) {/\* tu was \*/} |  |

#include <iostream>

#include <cmath>

*using* std::cout;

*using* std::cin;

*using* std::endl;

int main()

{

double radius, umfang;

char anfangVorname, anfangNachname;

*//* *Umfang-Berechnung*

cout << "Eingabe Radius:";

cin >> radius;

umfang = 2 \* M\_PI \* radius;

cout << "Der Kreis hat den Umfang: "

<< umfang << endl;

*//* *Anfangsbuchstaben* *von* *Vorname*

*//* *und* *Nachname*

cout << "Eingabe Vorname blank Nachname:";

cin >> anfangVorname;

cin.ignore(20, ’ ’);

cin >> anfangNachname;

cout << "Ihre Initialien lauten: "

<< anfangVorname

<< " und "

<< anfangNachname

<< endl;

*return* 0;

}

## Formatierte Ausgabe

|  |  |  |
| --- | --- | --- |
| **#include <iomanip>** | | |
| **Name** | **Anwendung** | **Wirkung** |
| *setw(int w)* | Ein-/Ausgabe | Setzen der Feldbreite auf w Stellen(entspricht width()) |
| *setprecision(int p)* | Ausgabe | Setzen der Genauigkeit auf p Stellen (entspricht precision()) |
| *setfill(int c)* | Ausgabe | Setzen des Fullzeichens auf ¨ das Zeichen c(entspricht fill()) |
| *setbase(int b)* | Ausgabe | Setzen der Basis des Zahlensystems auf b (nur fur b = 8, 10, 16) ¨ |
| *setiosflags(fmtflags f)* | Ein-/Ausgabe | Setzen der in f spezifizierten Formatflags (entspricht setf()) |
| *resetiosflags(fmtflags f)* | Ein-/Ausgabe | Rucksetzen der in ¨ f spezifizierten Formatflags (entspricht unsetf()) |

|  |  |  |
| --- | --- | --- |
| ***Name*** | **Anwendung** | **Wirkung** |
| *endl* | Ausgabe | Newline ausgeben und Puffer leeren |
| *ends* | Ausgabe | NULL-Char. ausgeben und Puffer leeren |
| *flush* | Ausgabe | Puffer leeren |
| *ws* | Eingabe | Whitespace-Character uberlesen ¨ |
| *dec* | Ein-/ Ausgabe | dezimale Darstellung ganzer Zahlen |
| *hex* | Ein-/ Ausgabe | hexadezimale Darstellung ganzer Zahlen |
| *oct* | Ein-/ Ausgabe | oktale Darstellung ganzer Zahlen |
| *left* | Ausgabe | linksbundige Ausgabe einstellen ¨ |
| *right* | Ausgabe | rechtsbundige Ausgabe einstellen ¨ |
| *internal* | Ausgabe | VZ linksbundig, Wert rechtsb ¨ undig ¨ |
| *fixed* | Ausgabe | Dezimalbruchdarstellung einstellen |
| *scientific* | Ausgabe | Exponentialdarstellung einstellen |
| *boolalpha* | Ausgabe | Ausgabe von Bool-Werten als Text, ansonsten als 0/1 |
| *noboolalpha* | Ausgabe | ios::boolalpha rucksetzen |
| *showpos* | Ausgabe | explizite Ausgabe eines pos. Vorzeichens |
| *noshowpos* | Ausgabe | ios::showpos r¨ ucksetzen |
| *uppercase* | Ausgabe | Ausgabe von Großbuchstaben bei hexadezimalen Zahlen und Gleitpunktzahlen in Exponentialdarstellung |
| *nouppercase* | Ausgabe | ios::uppercase rucksetzen |
| *showbase* | Ausgabe | Ausgabe mit Zahlensystem-Kennung (0 fur oktal, 0x f ¨ ur hexadezimal) |
| *noshowbase* | Ausgabe | ios::showbase rucksetzen |
| *showpoint* | Ausgabe | immer Ausgabe des Dezimalpunktes und abschließenden Nullen (bei Gleitpunktzahlen) |
| *noshowpoint* | Ausgabe | ios::showpoint rucksetzen |
| *skipws* | Eingabe | Uberlesen von f ¨ uhrenden Whitespaces |
| *noskipws* | Eingabe | ios::skipws rucksetzen |

### Beispiel Formatierte Ausgabe

#include <iostream>

#include <iomanip> *//* *fuer* *Manipulatoren*

*//* *mit* *Parameter*

*using* std::cout;

*using* std::endl;

int main()

{

*//* *Fuer* *ganze* *Zahlen*

int i = 255;

*//* *Umschalten* *auf* *hexadezimale* *Ausgabe*

*//* *Umschalten* *auf* *Ausgabe* *der* *Zahlensystem-Kennung*

cout << "i = " << std::hex << std::showbase << i << endl;

*//* *i* *=* *0xff*

*//* *Umschalten* *auf* *Gorssbuchstaben*

cout << "i = " << std::uppercase << i << endl;

*//* *i* *=* *0XFF*

*//* *Umschalten* *auf* *dezimale* *Ausgabe*

cout << "i = " << std::dec << i << endl;

*//* *i* *=* *255*

*//* *Fuer* *Gleitpunktzahlen*

double x = 4712.123456;

cout << "x = " << x << endl;

x = 4712.12

*//* *Umschalten* *auf* *Wissenschaftliche* *Darstellung*

cout << "x = " << std::scientific << x << endl;

*//* *x* *=* *4.712123E+03*

*//* *Setzen* *der* *Feld-Breite*

cout << "x = " << std::setw(20) << x << endl;

*//* *x* *=* *4.712123E+03*

*//* *Linksbuendige* *Ausgabe:*

cout << "x = " << std::setw(20)

<< std::left << x << endl;

*//* *x* *=* *4.712123E+03*

*//* *Setzen* *der* *Genauigkeit* *auf* *3* *Nachkommastellen*

cout << "x = " << std::setw(20)

<< std::setprecision(3) << x << endl;

*//* *x* *=* *4.712E+03*

*//* *Zureucksetzen* *auf* *Rechtsbuendige* *Ausgabe*

*//* *Setze* *?* *als* *Fuellzeichen*

cout << "x = " << std::setw(20) << std::right

<< std::setfill(’?’)

<< x << endl;

*//* *x* *=* *???????????4.712E+03*

*return* 0;

}

#include <iostream>

#include <iomanip>

void **seite77**()

{

*constexpr* int n = 5;

int num\_reads = 0;

double nettopreise[n];

*while* (num\_reads < n)

{

std::cin >> nettopreise[num\_reads];

++num\_reads;

}

*//* *print* *out* *table*

std::cout << "N | Netto| Brutto" << std::endl;

*for* (int i = 0; i < n; ++i)

{

std::cout << std::fixed;

std::cout << std::setw(4) << std::left << i << "|";

std::cout << std::setw(8) << std::setprecision(2)\

<< std::right << nettopreise[i] << "|";

std::cout << std::setw(10) << std::setprecision(2)\

<< std::right << nettopreise[i] \* 1.19 << std::endl;

}

}

## Referenzen





* Eine als Referenz deklarierte Variable ist ein zweiter Name(Alias)
* Operator& links vom Zuweisungs-Operator
* Operator & als Adressoperator verwendet steht er in der Regel rechts vom Zuweisungsoperator
* Der Typ der Referenz muss zum Typ des Objektes passen, auf das die Referenz verweist

Call by reference:

* Dabei werden die Werte nicht kopiert, sondern die Parameter sind ein zweiter Name!
* Jede Anderung innerhalb der Funktion am Parameter wirkt sich auf den Aktual-Parameter aus!
* Vorteil: Zum schreibenden (verandernden) Zugriff auf die Aktual-Parameter mussen keine Zeiger mehr ubergeben werden.

### Beispiel SWAP call by reference

void **swapR**(int& a, int& b) {

int hilf = a;

a = b;

b = hilf;

}

void **seite78**()

{

int a = 5, b = 8;

swapR(*a*, *b*);

std::cout << "a = " << a << ", b = " << b << std::endl;

}

### Beispiel Array (max, min, avg)

#include <cstddef>

#include <cmath>

#include <iostream>

*template*<std::size\_t n>

void **minMaxMittel**(double (&vec)[n], double& min, double& max, double& mittel) {

min = INFINITY;

max = -INFINITY;

mittel = NAN;

double sum = 0;

*for* (double d : vec) {

*if* (d > max) max = d;

*if* (d < min) min = d;

sum += d;

}

mittel = sum / n;

}

### Vergleich Zeiger/Referenzen

|  |  |
| --- | --- |
| **Zeiger** | **Referenz** |
| Eigenstandiges Objekt im Speicher | nur Aliasname fur anderes  programmiersprachliches  Objekt |
| hat eigene Adresse | hat keine eigene Adresse |
| kann uninitialisiert sein | muß immer initialisiert sein |
| zeigt nicht immer auf ein Objekt (z.B. NULL-Zeiger nullptr) | verweisen immer auf ein  Objekt |
| Zeiger auf Zeiger sind moglich | Referenz kann nicht auf  andere Referenz verweisen |

## Const Zeiger



### Zeiger auf const (Read-only-Zeiger)

 oder 

* Zeiger ptr zeigt auf einen konstanten Wert

(Read-Only-Zeiger)

* Zuweisung an ptr ist erlaubt, d. h. der Zeiger selber kann umgesetzt werden
* Zuweisung an \*ptr ist nicht erlaubt

### Konstante Zeiger



* Zuweisung an ptr ist nicht erlaubt, d. h. Zeiger kann im Verlauf auf keine andere Speicherposition zeigen.
* Zuweisung an \*ptr ist erlaubt, d. h. der Inhalt der Speicherposition, auf die ptr zeigt, kann verandert werden.

### Konstanter Read Only Zeiger



* Zeigervariable und Variable, worauf Zeiger zeigt, sind konstant
* Zuweisung an ptr ist nicht erlaubt
* Zuweisung an \*ptr ist nicht erlaubt.

### Beispiel const Zeiger

void **seite88**()

{

*//* *auskommentierte* *Zeilen* *führen* *zu* *Fehlern*

int i = -1;

*const* int ic = i;

*const* int\* pic = &ic;

*//FEHLER!!!!* *const* *zeiger* *auf* *const* *variable*

int\* *const* cpi = &ic;

*const* int\* *const* cpic = &ic;

i = ic;

pic = &ic;

*//Fehler!!!* *cpi* *deklaration* *hat* *vorhin* *nicht* *fktioniert*

cpi = pic;

pic = cpic;

*//Fehler!!!* *const* *zeiger* *mit* *read-only-var*

*//weder* *zuweisen* *noch* *wert* *veränderbar*

cpic = &ic;

ic = \*cpic;

}

## Const Referenz



* Referenz ref ist schreibgeschutzt (Read only).
* Uber eine konstante Referenz ist keine Veranderung des Inhaltes moglich.
* Eine Referenz, die auf ein konstantes Objekt verweist, muss selbst als konstant deklariert werden.
* Eine als konstant deklarierte Referenz darf auf ein nicht konstantes Objekt verweisen.
* Hauptsachliche Verwendung bei der Parameter-Ubergabe.

### Fehler bei const pointer/referencen

void **f1**(*const* double& x)

{

cout << x; *//lesender* *Zugriff*

}

void **f2**(double& y)

{

y = 1.2345; *//schreibender* *Zugriff*

}

double var = 0;

*const* double PI = 3.1415;

f1( var ); *//* *OK,* *Uebergabe* *an* *const* *erlaubt*

f1( PI ); *//* *OK,* *f1* *aendert* *nicht*

f2( var ); *//* *OK*

f2( PI ); *//* *Syntaxfehler,* *da* *f2* *aendert*

f2( 42.0 ); *//* *Syntaxfehler,* *da* *f2* *aendert*

int ival1 = 10;

int ival2 = 11;

int\* iPtr = &ival1;

int& iRef = ival2;

int\*& refPtr = iPtr; *//* *OK,* *Referenz* *auf* *einen*

*//* *Pointer*

cout << "iPtr = " << iPtr

<< ", refPtr = " << refPtr << endl;

iPtr = &iRef; *//* *OK,* *iPtr* *zeigt* *nun* *auf* *ival2*

cout << "iPtr = " << iPtr

<< ", &ival2 = " << &ival2 << endl;

int\*& refPtr2 = &ival1; *//* *Fehler,* *&ival1* *ist* *eine*

*//* *constante* *Adresse,* *hierauf*

*//* *kann* *nur* *eine* *Referenz* *auf* *einen*

*//* *konstanten* *Pointer* *verweisen*

int\**const*& refPtr3 = &ival1; *//* *OK,* *refPtr3* *ist* *Referenz*

*//* *auf* *die* *Anfangsadresse*

*//* *von* *ival1*

cout << "refPtr3 = " << refPtr3

<< ", &ival1 = " << &ival1 << endl;

int&\* ptrRef = &iRef; *//* *Fehler,* *es* *gibt* *keine* *Pointer*

*//* *auf* *Referenzen*

*const* int cival = 10;

int\*& refPtr4 = &cival; *//* *Fehler* *aus* *2* *Gruenden:*

*//* *es* *wird* *auf* *eine*

*//* *konstante* *Adresse*

*//* *mit* *einem* *konstanten* *Wert*

*//* *verwiesen!*

*const* int\*& refPtr4 = &cival; *//* *Fehler:* *es* *wird* *mit*

*//* *einer* *Refrerenz*

*//* *vom* *Typ*

*//* *nicht* *konstanter*

*//* *Pointer* *verwiesen*

int\**const*& refPtr5 = &cival; *//* *Fehler* *es* *wird* *mit*

*//* *einer* *Refrenz*

*//* *vom* *Typ*

*//* *konstanter* *Pointer,*

*//* *der* *aber* *nicht*

*//* *Read-Only* *ist*

*//* *verwiesen*

*const* int\**const*& refPtr6 = &cival; *//* *OK* *refPtr6*

*//* *ist* *Referenz* *auf*

*//* *die* *konstante*

*//* *Adresse* *von*

*//* *cival,* *deren*

*//* *Inhalt* *nicht*

*//* *veraendert*

*//* *werden* *darf*

cout << "refPtr6 = " << refPtr6

<< ", &cival =" << &cival << endl;

## Dynamische Speicher

### 1.6.1 Normal

**Anlegen (new)**



* Reserviert Speicher auf dem Heap fur einen Wert vom Typ ¨ type.

int\* ip1 = *new* int;

*struct* **Complex**

{

double re, im;

} z1 = {1.0, 2.0};

Complex\* pc1 = *new* Complex( z1 );

Complex\* pc2 = *new* Complex{ z1 };

**Freigeben (delete)**

* Das Objekt, auf den der Zeiger ptr zeigt wird zerst¨ ort und der reservierte dynamische Speicher wird freigegeben.
* Der Zeiger ptr muß auf Speicher zeigen, der zuvor mit new reserviert wurde

*delete* ip1; ip1 = *nullptr*;

*delete* ip2; ip2 = *nullptr*;

*delete* ip3; ip3 = *nullptr*;

*delete* pc1; pc1 = *nullptr*;

*delete* pc2; pc2 = *nullptr*;

### 1.6.2 Array

**Anlegen (new)**



* Es wird Speicher fur ein Feld mit anzahl Elementen vom Typ elementtyp reserviert
* new[] liefert die Adresse des ersten Feldelementes zuruck.

**Freigeben (delete)**



* Gibt den Speicherbereich frei, auf den ptr zeigt
* Dieser Speicherbereich muss mit new[] angelegt worden sein.

### 1.6.3 Beispiel dynamische Speicher

*//* *Allokation* *eines* *Vektors*

*constexpr* int N = 3;

double\* pvec = *new* double[N]{1.0, 2.0, 3.0};

*for*( int i = 0; i < N; ++i)

{

cout << pvec[i] << endl; *//* *Alternative* *fuer* *pvec[i]?*

}

*//* *Loeschen:*

*delete*[] pvec;

pvec = *nullptr*;

*//* *Allokation* *einer* *NxN* *Matrix:*

*//* *wobei* *N* *zur* *Kompilezeit* *bekannt* *ist*

*//* *double\*\** *mat* *=* *new* *double[N][N];* *//* *Syntax* *Fehler!!!!*

*//* *Statt* *dessen:*

*//* *mat* *ist* *ein* *Pointer* *auf* *ein*

*//* *Feld* *mit* *N* *Elementen*

*//* *welche* *der* *beiden* *Anzahlen* *in* *double[N][N]*

*//* *kann* *durch* *eine* *andere* *Konstante* *ersetzt* *werden?*

double (\*mat)[N] = *new* double[N][N];

int k = 1;

*for*( int i = 0; i < N; ++i){

*for*(int j = 0; j < N; ++j){

mat[i][j] = k++;

}

}

*for*( int i = 0; i < N; ++i){

*for*( int j = 0; j < N; ++j){

cout << std::setw(12) << std::setprecision(2)

<< mat[i][j]; *//* *Alternative* *fuer* *mat[i][j]?*

}

cout << endl;

}

*delete*[] mat; mat = *nullptr*;

*//* *Anlegen* *einer* *mxn* *Matrix*

*//* *m,* *n* *werden* *eingelesen*

int m, n;

cout << "Eingabe m und n: ";

cin >> m >> n;

cout << m << ", " << n << endl;

*//* *Anlegen* *eines* *Feldes* *mit* *m* *Pointern* *auf* *double*

double\*\* mat2 = *new* double\*[m];

*//* *Anlegen* *der* *Zeilenvektoren* *und* *fuellen*

k = 1;

*for*(int i = 0; i < m; ++i){

mat2[i] = *new* double[n];

*for*(int j = 0; j < n; ++j) {

mat2[i][j] = k++;

}

}

*for*( int i = 0; i < m; ++i){

*for*( int j = 0; j < n; ++j){

cout << std::setw(12) << std::setprecision(2)

<< mat2[i][j];

}

cout << endl;

}

*//* *Speicher* *freigeben* *in* *umgekehrter* *Reihenfolge*

*//* *Loeschen* *der* *Zeilen-Vektoren*

*for*(int i = 0; i < m; ++i) {

*delete*[] mat2[i]; mat2[i] = *nullptr*;

}

*//* *Loeschen* *des* *Feldes* *mit* *den* *Pointer-Elementen*

*delete*[] mat2; mat2 = *nullptr*;

## Funktionszeiger



* Funktionszeiger sind Zeiger, die auf die Speicheradresse einer Funktion zeigen.
* Ein Funktionszeiger kann an andere Funktion ubergeben werden oder in Datenstrukturen gespeichert werden

int **add**(int x, int y)

{

*return* x + y;

}

int **mult**(int x, int y)

{

*return* x \* y;

}

int main() {

*//Funktionszeiger* *op* *deklarieren*

int (\*op)(int, int);

op = add; *//* *oder:* *op* *=* *&add;*

cout << op(3,4) << endl; *//* *Aufruf* *add*

op = mult;

cout << op(3,4) << endl; *//* *Aufruf* *mult*

*return* 0;

}

### const Funktionszeiger

* Benotigt man in der Funktion nur Lesezugriff, ist es Pflicht die Parameter mittels const vor einer unbeabsichtigten Anderung zu sch ¨ utzen (const correctness) .
* In einem derartigen Fall ist immer die Ubergabe als ¨ const Referenz oder const Pointer einer call by value Wertubergabe ¨ vorzuziehen.
  + void druckeAuto(const Auto& a) { /\*...\*/ }
  + void druckeAuto(const Auto\* a) { /\*...\*/ }

## Default Parameter



* void func(double r, int a=0, int b=1);

|  |  |
| --- | --- |
| Zulässig: | Unzulässig |
|  |  |

## Inline Funktion



* Ziel: Vermeidung des Aufruf-Overheads.
* Die inline Definition ist eine Empfehlung an den Compiler, den Funktionsaufruf durch den Code des Funktionsrumpfes direkt zu ersetzen, d.h. die Funktion wie ein Makro zu expandieren
* Sinnvoll ist die Inline-Expandierung i. A. nur bei sehr kurzen Funktionen
* Inline-Funktionen können nicht in andere Module exportiert werden, also nicht die Speicherklasse extern

## Typplatzhalter auto

* Der Typplatzhalter auto kann immer dann verwendet werden, wenn der Datentyp implizit festgelegt ist, z. B. durch den Initialisierungswert.

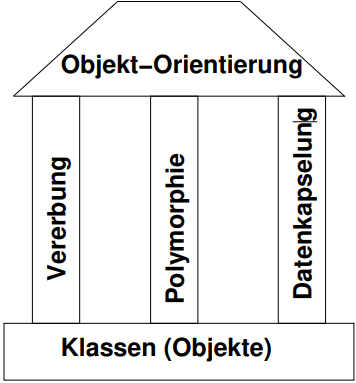
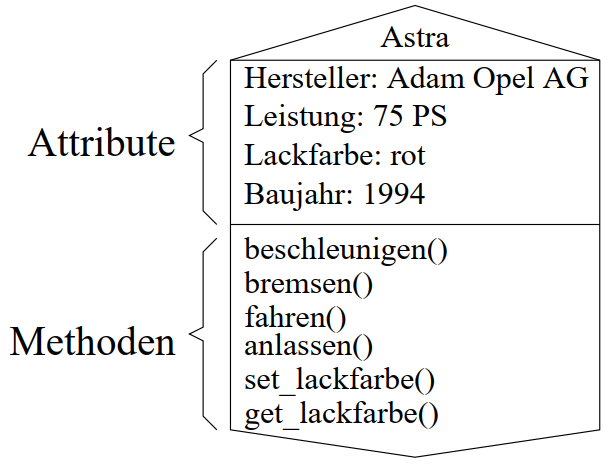
|  |  |
| --- | --- |
| Zulässig | Unzulässig |
|  |  |

## Bereichbasierte for schleife

* Wie Python mit *for i in list*

|  |  |
| --- | --- |
|  | int werte[10] {1, 2, 3, 4, 5,  6, 7, 8, 9, 10};  *auto* summe = 0;  *for* (*auto* element : werte)  {  summe += element;  }  cout << "Summe = " << summe << endl; |
| std::vector<int> vec = {1, 2, 3, 4, 5};  *for*(*auto* i: vec)  {  cout << i << endl;  } | *for*( std::vector<int>::iterator it = vec.begin();it != vec.end(); ++it)  {  cout << \*it << endl;  } |
| *for*(*auto*& i: vec){  i = i\*i;  }  *for*( *auto* *i*: vec){  cout << i << endl;  } | *for*(*auto* it = vec.begin(); it != vec.end(); ++it)  {  cout << \*it << endl;  } |

# Objektorientiertes Programmieren



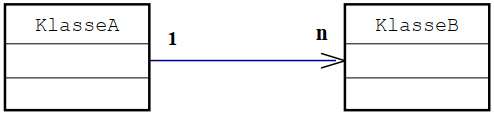
**UML:**

|  |  |
| --- | --- |
| **Modifikator** | **Bedeutung** |
| *public* | offentlich: Zugriff ist uberall erlaubt |
| *private* | privat: Zugriff ist nur innerhalb der Klasse erlaubt |
| *protected* | geschutzt: Zugriff ist in der Klasse ¨ und in Unterklassen erlaubt |

## Assosation; Agregation

**Assosation:**

* Eine Assoziation beschreibt eine Beziehung zwischen Klassen.
* mechanikerMike.wartet(lkw23);
* In UML-Notation wird eine Assoziation durch eine Verbindungslinie

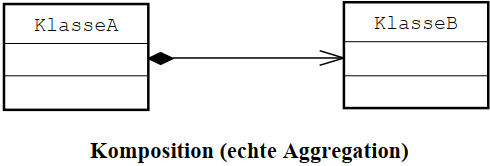
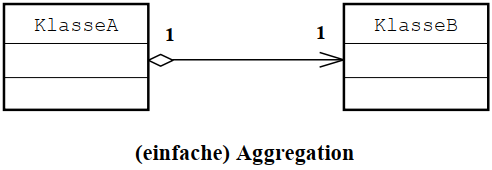
zwischen den Klassen dargestellt.

* Die Multiplizit¨at gibt an wie viel Objekte aus beiden Klassen

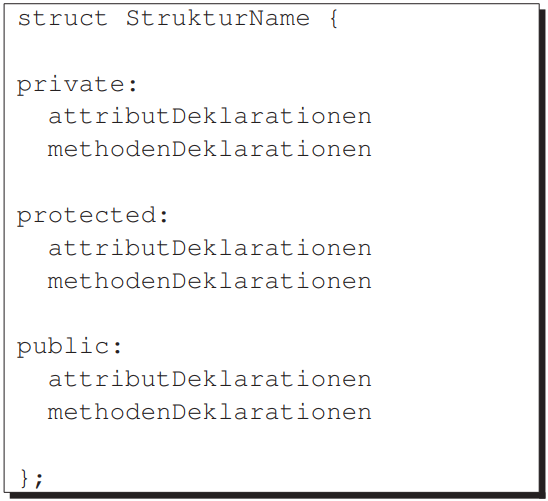
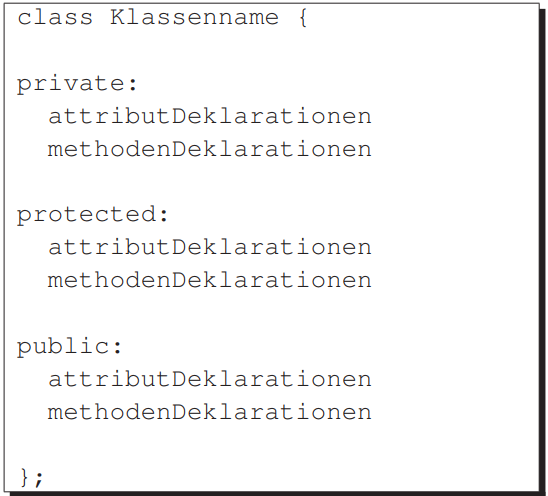
miteinander in Beziehung stehen. Gultige Notationen sind z.B.: ¨ 3, 0..7, \*, 2..\*.

**Agregation**

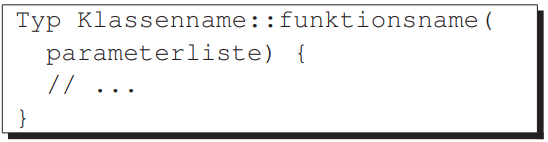
* Eine spezielle Form der Assoziation ist die Aggregation. Sie liegt vor wenn eine Klasse (das Aggregat) Objekte einer anderen Klasse als Bestandteile hat. Es besteht eine Teil-Ganzes-Beziehung (has a), die im UML-Diagramm durch eine offene Raute am Aggregat kenntlich gemacht wird.



## Definition von Klassen/Strukt



## Memberfunktionen (Implementierung in CPP-Datei)



|  |  |
| --- | --- |
| **Header-Datei** | **MemberFunktion-CPP-Datei** |
| #ifndef DATUM\_H  #define DATUM\_H  *class* **Datum** {  *private*:  *//* *Attribute* *der* *Klasse*  int jahr, monat, tag;  *public*:  *//* *Member-Funktionen* *(Methoden)*  void **setDatum**(int j, int m, int t);  void **printDatum**() *const*;  };  #endif *//* *DATUM\_H* | #include "Datum.h"  void Datum::**setDatum**(int j, int m, int t)  {  jahr = j;  monat = m;  tag = t;  }  void Datum::**printDatum**() *const*  {  std::cout << tag << "." << monat << "."  << jahr << std::endl;  } |

### const Memberfunktion



* Dadurch wird sichergestellt, dass der Zustand des aktuellen Objektes nicht innerhalb der konstanten Member-Funktion versehentlich verandert werden kann
* Beim Aufruf einer konstanten Member-Funktion wird dann der verborgene konstante Pointer this auf das aktuelle Objekt sogar als Read-Only Pointer ubergeben:



* Innerhalb einer konstanten Member-Funktion ist es nicht moglich, nicht konstante Member-Funktionen aufzurufen.

### Beispiel Memberfunktion Datum

|  |  |
| --- | --- |
| **Header** | **CPP** |
| #ifndef DATUM\_H  #define DATUM\_H  *class* **Datum** {  *private*:  *//* *statische* *Klassenvariablen*  *static* *const* int jahrDefault = 1970,  monatDefault = 1,  tagDefault = 1;  *//* *private* *statische* *Hilfs-Methoden*  *static* bool **checkTag**(int m, int t);  *//* *private* *statische* *inline* *Hilfs-Methode*  *static* bool **checkDatum**(int j, int m, int t)  {  *return* j >= 0 && 1 <= m && m <= 12 && checkTag(m, t);  }  *//* *Attribute*  int jahr, monat, tag;  *//* *private* *Member-Funktion*  void **setDefault**();  *public*:  *//* *jetzt* *als* *inline* *Methode*  void **setDatum**(int j, int m, int t)  {  *if* (checkDatum(j, m, t))  {  jahr = j;  monat = m;  tag = t;  }  *else*  {  setDefault();  }  }  *//* *...*  };  #endif *//* *DATUM\_H* | #include "datum.h"  #include <iostream>  *const* int Datum::jahrDefault = 1970,  Datum::monatDefault = 1,  Datum::tagDefault = 1;  bool Datum::**checkTag**(int m, int t)  {  bool erg = *false*;  *switch* (m)  {  *case* 2:  *//* *Schaltjahre* *werden* *ignoriert*  erg = (1 <= t && t <= 28);  *break*;  *case* 4:  *case* 6:  *case* 9:  *case* 11:  erg = (1 <= t && t <= 30);  *break*;  *default*:  erg = (1 <= t && t <= 31);  *break*;  }  *return* erg && (t > 0);  }  void Datum::printDatum() *const*  {  std::cout << tag << "." << monat  << "." << jahr << std::endl;  }  void Datum::**setDefault**()  {  jahr = jahrDefault;  monat = monatDefault;  tag = tagDefault;  } |

## inline Methoden



* Ziel: Vermeidung des Aufruf-Overheads.
* Die inline Definition ist eine Empfehlung an den Compiler, den Funktionsaufruf durch den Code des Funktionsrumpfes direkt zu ersetzen, d.h. die Funktion wie ein Makro zu expandieren.
* An jeder Aufrufstelle einer Inline-Funktion kann der Compiler den vollst¨andigen Maschinencode des Funktionsrumpfes anstelle des Funktionsaufrufs einfugen.
* Sinnvoll ist die Inline-Expandierung i.A. nur bei sehr kurzen Funktionen

## Instanzen (Speicher statisch, dynamisch)

|  |  |
| --- | --- |
| **Statisch** | **Dynmaisch** |
| Punkt-Notation | Pfeil-Notation oder (\*p) Punkt-Notation |
|  |  |

#include <iostream>

#include "datum.h"

int main() {

Datum heute;

heute.printDatum();

heute.setDatum(2019, 10, 18);

heute.printDatum(); *//* *Ausgabe:* *18.10.2019*

*//* *Dynamische* *Instanz* *(Heap)*

Datum\* pd = *new* Datum;

(\*pd).setDatum(1900, 1, 1);

pd->printDatum();

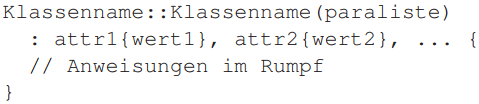
*delete* pd;

pd = *nullptr*;

*return* 0;

}

## Konstruktor: Initialisierungsliste + Default



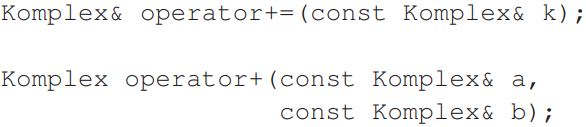
|  |  |
| --- | --- |
| *class* **Datum** {  *private*:  int jahr, monat, tag;  *public*:  **Datum**(int j, int m, int t)  : jahr{j}, monat{m}, tag{t} { *//....}*  }; | Datum d911{2001, 9, 11};  d911.printDatum(); |
| *class* **Datum** {  *private*:  int jahr, monat, tag;  *public*:  *//* *Konvertierender* *Konstruktor*  **Datum**(int j)  : jahr{j}, monat{1}, tag{1} {}  *//* *...*  }; | *class* **Datum** {  *private*:  int jahr, monat, tag;  *public*:  *//* *Default-Konstruktor* *als*  *//* *delegierender* *Konstruktor*  **Datum**() : Datum(2000, 1, 1) {}  **Datum**(int j, int m, int t)  : jahr{j}, monat{m}, tag{t} {*//...}*  }; |
| *class* **Datum** {  *private*:  int jahr, monat, tag;  *public*:  *//* *Konstruktor* *mit* *Default-Parametern*  *//* *es* *muss* */darf* *zusaetzlich* *kein*  *//* *Default-Konstruktor* *definiert* *werden*  *explicit* **Datum**(int j=2000, int m=1, int t=1)  : jahr{j}, monat{m}, tag{t} { *//....* *}*  *//....*  }; | *//* *Bsp.* *Konvertierender* *Konstruktor*  *class* **Datum** {  *private*:  int jahr, monat, tag;  *public*:  *//* *Konvertierender* *Konstruktor*  **Datum**(int j)  : jahr{j}, monat{1}, tag{1} {}  *//* *...*  }; |

# Überladung von Operatoren

Mögliche Operatoren:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| -> | [] | () | ++ | -- |
| ! | ˜ | \* (un¨ar) | + (un¨ar) | - (un¨ar) |
| & (un¨ar) | new | delete | -> |  |
| / | % | \* (bin¨ar) | + (bin¨ar) | - (bin¨ar) |
| << | >> | < | <= | > |
| >= | == | != | & (bin¨ar) | ˆ |
| | | && | || | = | \*= |
| /= | %= | += | -= | <<= |
| >>= | &= | ˆ= | |= | , |





## Ausgabe-Operator

|  |
| --- |
| *class* **Komplex** {  *private*:  double\* pk;  *public*:  void **output**(std::ostream& os) *const*  {  os << "(" << pk[0] << ", " << pk[1] << ")";  }  };  *//* *Ueberladen* *des* *Ausgabe-Operators* *als*  *//* *freie* *Funktion*  *inline* std::ostream& *operator*<<(std::ostream& os, *const* Komplex& z) {  z.output(*os*);  *return* os;  } |

## [] Operator

* Immer als Referenz
  + double& operator[]

#include <stdexcept>

*class* **Komplex** {

*private*:

double\* pk;

*public*:

double& *operator*[](int i) {

*if* (0 <= i && i <= 1) {

*return* pk[i];

}

*else* {

*throw* std::out\_of\_range("ERROR: falscher Index");

}

}

*const* double& *operator*[](int i) *const* {

*if* (0 <= i && i <= 1) {

*return* pk[i];

}

*else*{

*throw* std::out\_of\_range("ERROR: falscher Index");

}

}

};

## Weitere Operatoren

|  |
| --- |
| *class* **Komplex** {  *private*:  double\* pk;  *public*:  **Komplex**() : pk{*new* double[2]{0, 0}} {}  *explicit* **Komplex**(double re, double im)  : pk{*new* double[2]{re, im}} {}  Komplex& *operator*+=(*const* Komplex& a) {  pk[0] += a.pk[0];  pk[1] += a.pk[1];  *return* \**this*;  }  Komplex& *operator*+=(*const* double a) {  pk[0] += a;  *return* \**this*;  }  };  Komplex *operator*+(*const* Komplex& a, *const* Komplex& b);  Komplex *operator*+(*const* Komplex& a, double b);  Komplex *operator*+(double a, *const* Komplex& b); |
| Komplex *operator*+(*const* Komplex& a,*const* Komplex& b) {  Komplex sum = a;  sum += b;  *return* sum;  }  Komplex *operator*+(*const* Komplex& a, double b) {  Komplex sum{a[0] + b, a[1]};  *return* sum;  }  Komplex *operator*+(double a, *const* Komplex& b) {  Komplex sum{a + b[0], b[1]};  *return* sum;  } |
| *class* **Komplex** {  *private*:  double\* pk;  *public*:  **Komplex**(double re=0, double im=0)  : pk{*new* double[2]{re, im}} {}  Komplex& *operator*+=(*const* Komplex& a) {  pk[0] += a.pk[0];  pk[1] += a.pk[1];  *return* \**this*;  }  };  *inline* Komplex *operator*+(*const* Komplex& a,*const* Komplex& b) {  Komplex sum = a; sum += b;  *return* sum;  } |

## Präfix/Postfix

|  |  |
| --- | --- |
| *class* **Komplex** {  *private*:  double\* pk;  *public*:  *//* *............*  *//* *Praefix-Form*  Komplex& *operator*++() {  ++pk[0];  ++pk[1];  *return* \**this*;  }  *//* *Postfix-Form*  Komplex *operator*++(int dummy) {  Komplex tmp{\**this*};  ++\**this*;  *return* tmp;  }  }; |  |

## The Big 3

|  |  |  |
| --- | --- | --- |
| **Copy-Konstruktor** | **Copy-Zuweisungsoperator** | **Destruktor** |
|  |  |  |

|  |  |
| --- | --- |
| *class* **Komplex** {  *private*:  double\* pk;  *public*:  *//* *...*  *//* *Copy* *Konstruktor,* *erstellt* *tiefe* *Kopie*  **Komplex**(*const* Komplex& rhs) {  pk = *new* double[2];  pk[0] = rhs.pk[0];  pk[1] = rhs.pk[1];  }  *//* *Copy* *Zuweisungs-Operator,* *erstellt* *tiefe* *Kopie*  Komplex& *operator*=(*const* Komplex& rhs) {  *if* (*this* != &rhs) {  *delete*[] pk;  pk = *new* double[2];  pk[0] = rhs.pk[0];  pk[1] = rhs.pk[1];  }  *return* \**this*;  }  }; | int main() {  int i;  Komplex z1{1, 2}, z2;  Komplex z3 = z1; *//* *Copy-Konstruktor*  z2 = ++z1; *//* *Copy-Zuweisung*  cout<<"z1: "<<z1<<endl; cout<<"z2: "<<z2<<endl;  cout << "z3: " << z3 << endl;  z2[1] = -42;  cout << "z1: "<<z1<<endl; cout<<"z2: "<<z2<<endl;  cout << "z3: " << z3 << endl;  cin >> i;  } |

# Vererbung



* Die Klasse C wird von den Basisklassen B1, B2, ..., Bn abgeleitet.
* Durch das Zugriffsattribut public wird der Zugriff auf dieMember der Basisklasse geregelt.
* Grundsatzlich gilt, dass die abgeleitete Klasse keinen direkten Zugriff auf die privaten Member der Basisklasse hat.
* public-Vererbung werden alle ¨ offentlichen Member aus der Basis-Klasse auch in der abgeleiteten Klasse ¨ offentlich gemacht.

|  |  |
| --- | --- |
| #include <iostream>  *class* Fahrzeug {  *private*:  int herstellungsJahr;  *public*:  Fahrzeug(int jahr) :herstellungsJahr{jahr} {}  void output(std::ostream& os) *const* {  os << "Fahrzeug, hergestellt im  Jahr: "  << herstellungsJahr  << std::endl;  }  ~Fahrzeug();  }; | *class* **KFZ** : *public* Fahrzeug {  *private*:  double ps;  *public*:  **KFZ**(int jahr, double p) : Fahrzeug{jahr}, ps{p} {}  void ***output***(std::ostream& os) *const* {  *//Aufruf* *Basisklassen-Meth.*  Fahrzeug::output(*os*);  os << "Spezieller Typ KFZ mit: "  << ps  << " PS" << std::endl;  }  ~**KFZ**();  }; |

# Polymorphie

|  |
| --- |
| double flaeche;  *if* (x.type() == "Kreis") {  flaeche = M\_PI \* x.getR()\*x.getR();  }  *if* (x.type() == "Rechteck") {  flaeche = x.getA() \* x.getB();  } |

## Cast Operatoren

|  |
| --- |
| C-Stil  double preis = 12.34;  int euro;  euro = (int)preis;  *//* *oder* *in* *der* *Konstruktor-Schreibweise*  euro = int(preis); |

### Static Cast

*class* **A** { *//* *Klasse* *mit* *einparametrigem*

*private*: *//* *Konvertierungs-Konstruktor*

int a;

*public*:

**A**(int i): a{i} {}

};

void **funcA**(*const* A& obj) {

cout << "Aufruf funcA" << endl;

}

*class* **Basis** {

*public*:

*virtual* void ***output***() *const* {cout << "Basis" << endl;}

};

*class* **Kind** : *public* Basis {

*public*:

*virtual* void ***output***() *const* {cout << "Kind" << endl;}

void **kindMethode**() *const* {cout << "Kind-Methode" << endl;}

};

int main() {

double preis = 12.34; int euro;

euro = *static\_cast*<int>(preis);

funcA(*static\_cast*<A>(euro));

Kind kind;

Basis\* basisPtr = &kind;

Kind\* kindPtr = *static\_cast*<Kind\*>(basisPtr);

kindPtr->kindMethode();

}

* Fur einen Down-Cast kann der static\_cast Operator nur verwendet werden, wenn zur Compile-Zeit feststeht, dass der Basisklassen-Zeiger auf ein Objekt einer abgeleiteten Klasse verweist

### Dynamic Cast



* Der Typ T muss ein Zeiger oder eine Referenz auf eine Klasse sein

int main() {

Kind k;

Nesthaekchen n;

Basis\* pb = &k;

pb->output();

*//* *Fehler* *ein* *Downcast* *ist* *notwendig*

*//* *pb->kindMethode();*

*dynamic\_cast*<Kind\*>(pb)->kindMethode();

*//* *falls* *cast* *nich* *Moeglich* *erhaelt* *man* *nullptr* *zurueck*

*if* (*dynamic\_cast*<Nesthaekchen\*>(pb) == *nullptr*) {

cout << "downcast zu Nesthaekchen nicht moeglich"

<< endl;

}

*//* *Falls* *bei* *Referenzen* *cast* *nicht* *moeglich*

*//* *wird* *eine* *bad\_cast* *Exception* *ausgeworfen*

Basis& rb = n;

Kind& rk = *dynamic\_cast*<Kind&>(rb);

rk.kindMethode();

*return* 0;

}

### Const Cast



* Dieser Operator ist der einzige Operator, mit dem die const-Eigenschaft eines Objektes entfernt werden kann
* Er wird benotigt, wenn fur eine Methode, die das eigene Objekt nicht verandert, die const Markierung vergessen wurde.

void funcVec(*const* SillyVec\* pcv) {

int s;

*//* *Fehler,* *mit* *const* *Pointer* *darf* *Methode,* *die*

*//* *keine* *const* *Markierung* *hat,* *NICHT* *aufgerufen* *werden*

*//* *s* *=* *pcv->getSize();*

*//* *deshalb* *muss* *const-Markierung* *von* *pcv* *entfernt* *werden*

SillyVec\* pv = *const\_cast*<SillyVec\*>(pcv);

s = pv->getSize();

cout << s << endl;

}

### Reinterpret Cast



* Dieser Cast-Operator kann die const-Eigenschaft eines Objektes nicht andern, aber ansonsten ist jede Typumwandlung moglich
* Die Verwendung dieses Cast-Operators ist ein sehr fragwurdiger Hack!!

int main() {

void\* p = std::malloc(10 \* *sizeof*(int));

int\* ip = *reinterpret\_cast*<int\*>(p);

\*ip = 1;

*return* 0;

}

### Beispiel Cast

#include <cmath>

#include <cstdlib>

#include <ctime>

#include <iostream>

*using* std::cout;

*using* std::endl;

*class* **X** {

*virtual* void ***output***() *const* {cout << "X" << endl; }

};

*class* Kreis : *public* X {

*private*:

double r;

*public*:

Kreis(double aR) : r{aR} {}

double getR() *const* { *return* r; }

};

*class* Rechteck : *public* X {

*private*:

double a, b;

*public*:

Rechteck(double aA, double aB) : a{aA}, b{aB} {}

double getA() *const* { *return* a; }

double getB() *const* { *return* b; }

};

int main() {

Kreis k{2};

Kreis\* kp;

Rechteck r{2, 3};

Rechteck\* rp;

X\* xp;

std::srand(*static\_cast*<unsigned int>(time(*nullptr*)));

double zufall = *static\_cast*<double>(rand()) / RAND\_MAX;

*if* (zufall >= 0.5) {

xp = &k;

} *else* {

xp = &r;

}

double flaeche;

kp = *dynamic\_cast*<Kreis\*>(xp);

rp = *dynamic\_cast*<Rechteck\*>(xp);

*if* (kp != *nullptr*) {

*//* *unbekanntes* *Objekt* *hat* *den* *Typ* *Kreis*

flaeche = M\_PI \* kp->getR()\*kp->getR();

}

*if* (rp != *nullptr*) {

*//* *unbekanntes* *Objekt* *hat* *den* *Typ* *Rechteck*

flaeche = rp->getA() \* rp->getB();

}

cout << "Flaeche = " << flaeche << endl;

*return* 0;

}

## Typeid



## virtual Methode



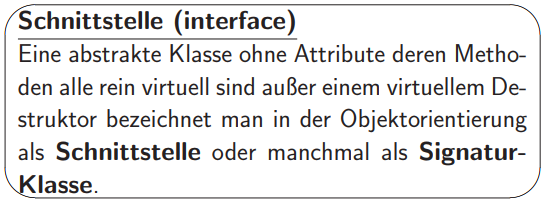
* Konstruktoren konnen nicht virtuell sein.
* Destruktoren konnen virtuell sein (s.u.).

### Abstrakte virtual Klassen

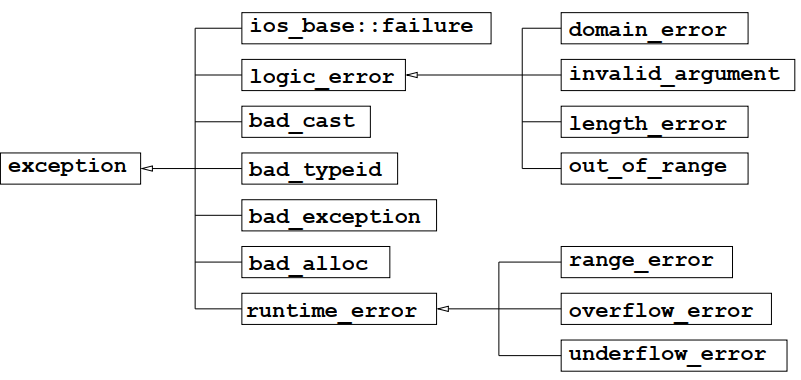


* Die Methode muss dann in dieser Klasse nicht mehr implementiert (also definiert) werden.
* Dies dient als Festlegung einer gemeinsamen Schnittstelle.
* Besitz eine Klasse mindestens eine rein virtuelle Methode, so konnen von dieser Klasse keine Objekte erstellt werden.

|  |  |
| --- | --- |
| **Abstrakte Klasse** | **Konkrete Klasse** |
| enthalt mindestens eine rein virtuelle Methode. | alle Methoden sindimplementiertjede geerbte rein virtuelle Methode muß implementiert sein |
| Es k¨ onnen keine Objekte der Klasse erstellt werden, allerdings k¨ onnen Pointer und Referenzen vom Typ der Klasse deklariert werden. | Das Instanzieren der Klasse istm¨ oglich. |
| Zweck: Festlegung einer gemeinsamen Schnittstelle | Zweck: Speicherung einesObjektes |

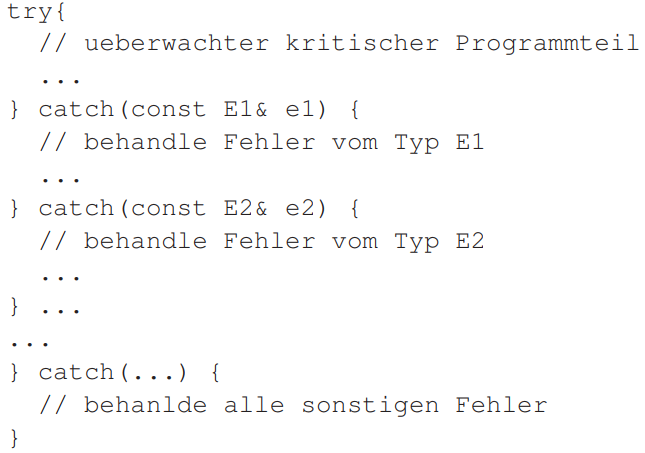
* Schnittstellen sind ein sehr wichtiges Mittel, um in großenProgrammen Module und Programm-Teile zu entkoppeln.
* Der Zweck einer Schnittstelle ist die Definition eines Vertrages (design by contract) zwischen Objekten

# Exceptions





## Try-cath-Block



#ifndef EXCEPTION\_H

#define EXCEPTION\_H

#include <iostream>

#include <stdexcept>

*using* std::cin;

*using* std::cout;

*using* std::endl;

*class* **MyException** : *public* std::out\_of\_range {

*private*:

int myErrorNumber;

*public*:

**MyException**(*const* char\* m, int eN)

: std::out\_of\_range(m), myErrorNumber{eN} {}

void **printMyMessage**() *const* {

cout << *what*() << " Fehler-Nummer: "

<< myErrorNumber << endl;

}

};

void **func**() {

int x, y;

bool done = *false*;

*//* *Ausnahme-Ausloesung* *fuer* *cin* *einschalten*

cin.exceptions(std::ios\_base::failbit);

*while* (!done) {

*try* {

cout << "Eingabe einer positive Zahl x: ";

cin >> x;

*if* (cin.fail()) {

*throw* std::runtime\_error("keine ganze Zahl");

}

*if* (x == 1) {

*throw* std::domain\_error(

"es droht eine Division durch Null");

}

*if* (x < 0) {

*throw* std::out\_of\_range("negativer Wert!");

}

*if* (x == 99) {

*throw* MyException("Eingabe von 99!!", 99);

}

y = 42 / (x - 1);

cout << y << endl;

*if* (y == 42) {

done = *true*;

}

} *catch* (*const* std::ios\_base::failure& e) {

*//* *cin* *zureuck* *setzen* *und* *Puffer* *leeren*

cin.clear();

*//* *fehlerhaftes* *Zeichen* *entfernen*

std::string trash;

cin >> trash;

cout << e.*what*() << endl;

cout << trash << " keine ganze Zahl!" << endl;

} *catch* (*const* MyException& e) {

e.printMyMessage();

*//* *Exception* *weiter* *leiten*

*throw*;

} *catch* (*const* std::runtime\_error& e) {

cout << e.*what*() << endl;

cin.clear();

*//* *fehlerhaftes* *Zeichen* *entfernen*

std::string trash;

cin >> trash;

} *catch* (*const* std::domain\_error& e) {

cout << e.*what*() << endl;

} *catch* (*const* std::out\_of\_range& e) {

cout << e.*what*() << endl;

} *catch* (...) {

*throw*;

}

}

}

int **main**() {

cout << "Exception-Demo" << endl;

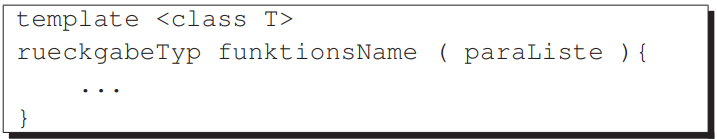
func();

*return* 0;

}

#endif *//* *EXCEPTION\_H*

# Templates



1. Anstelle von **template<class T>** kann man auch **template<typename T>** schreiben

*template* <*class* T>

T **max**(T a, T b) {

*return* a > b ? a : b;

}

int main() {

int i = 2, j = 3;

double x = 42.3, y = -2.9;

float f = -1.5f;

cout << max(i, j) << endl;

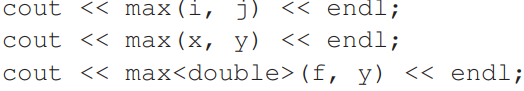
cout << max(x, y) << endl;

cout << max<double>(f, y) << endl;

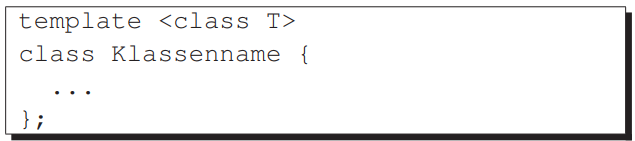
*return* 0;

}





## Klassentemplates

*template*<*class* T>

*class* **Set**{

*//* *hier* *ersetzt* *der* *Compiler*

*//* *Set* *durch* *Set<T>*

**Set**(*const* Set&);

*//* *der* *Typparameter* *T* *ist* *hier*

*//* *nicht* *notwendig* *s.o.*

Set<T>& *operator*=(*const* Set<T>&);};

|  |  |
| --- | --- |
| **Header** | **CPP** |
| *template* <*class* T>  *class* **Array** {  *private*:  int n;  T\* data;  void **copy\_data**(*const* Array& rhs) {  *for* (int i = 0; i < rhs.n; ++i) {  data[i] = rhs.data[i];  }  }  *public*:  **Array**(int n\_)  : n{n\_}, data{*new* T[n]} {}  **Array**(*const* Array& rhs) {  n = rhs.n;  data = *new* T[n];  copy\_data(rhs);  }  int **size**() *const* { *return* n; }  Array& *operator*=(*const* Array& rhs);  T& *operator*[](int i);  ~**Array**();  }; | #include <stdexcept>  *template* <*class* T>  Array<T>::~**Array**() {  *delete*[] data;  data = *nullptr*;  n = 0;  }  *template* <*class* T>  Array<T>& Array<T>::*operator*=(*const* Array& rhs) {  *if* (*this* != &rhs) {  n = rhs.n;  *delete*[] data;  data = *new* T[n];  copy\_data(rhs);  }  *return* \**this*;  }  *template* <*class* T>  T& Array<T>::*operator*[](int i) {  *if* (0 <= i && i < n) {  *return* data[i];  } *else* {  *throw* std::invalid\_argument("Ungueltiger Index");  }  }  *//* *Explizite* *Instanzierung*  *template* *class* **Array**<int>;  *template* *class* **Array**<double>;  *template* *class* **Array**<*const* char\*>; |
| **main** | |
| #include <iostream>  *using* std::cout;  *using* std::endl;  int **main3**() {  Array<int> vec{3};  *for* (int i = 0; i < vec.size(); ++i) {vec[i] = i;}  vec[1] = 42;  *for* (int i = 0; i < vec.size(); ++i) {cout << i << ", " << vec[i] << endl;}  Array<int> vec\_neu{4};  vec\_neu = vec;  *for* (int i = 0; i < vec\_neu.size(); ++i) {  cout << i << ", " << vec\_neu[i] << endl;  }  Array<*const* char\*> teilnehmer{2};  teilnehmer[0] = "Max";  teilnehmer[1] = "Hans";  *for* (int i = 0; i < teilnehmer.size(); ++i) {  cout << i << ", " << teilnehmer[i] << endl;  }  *return* 0;  } | |

# Beispiele

## Pascal Dreieck

#include <iostream>

void **pascalDreieck**(int n) {

int \*\*arrs = *new* int\*[n]; *//* *contains* *pointers* *to* *each* *line* *array*

*for*(int i = 0; i < n; ++i) {

arrs[i] = *new* int[i+1];

int\*& arr = arrs[i];

*for* (int j = 0; j < i+1; ++j) {

*//* *linker* *und* *rechter* *Rand* *ist* *immer* *eins*

*if* (j == 0 || j == i) {

arr[j] = 1;

} *else* {

*//* *add* *the* *two* *fields* *that* *are* *above*

arr[j] = arrs[i-1][j-1] + arrs[i-1][j];

}

}

}

*//* *Ausgabe*

*for* (int i = 0; i < n; ++i) {

*//* *Leerzeichen* *drucken*

*for*( int s = 0; s < (n-1) - i; ++s) {

std::cout << " ";

}

*//* *Zahlen* *drucken*

*for*( int j = 0; j < i+1; ++j) {

std::cout << arrs[i][j] << " ";

}

*//* *Neue* *Zeile*

std::cout << std::endl;

}

*//* *Ressourcen* *wieder* *freigeben*

*for*(int i = 0; i < n; ++i) {

*delete*[] arrs[i];

}

*delete*[] arrs;

}

void **seite105**() {

pascalDreieck(5);

pascalDreieck(10);

}

## Funktionszeiger Trapezregel

#include <cmath>

#include <iostream>

double **trapez**( double (\*f)(double), double a, double b, int n) {

double retval = NAN;

*if* (n>0) {

double h = (b-a)/n;

double sum = 0;

*for*(int j = 1; j < n; ++j) {

sum += f(a + j\*h);

}

retval = h/2 \* (f(a) + f(b) + 2\*sum);

}

*return* retval;

}

#include <cmath>

void **seite109**() {

*constexpr* int n = 15;

double res1[n-1], res2[n-1];

*for* (int i = 0; i < n-1; ++i) {

res1[i] = trapez([](double x){ *return* std::sin(x);}, 1, 2, i+1);

res2[i] = trapez([](double x){ *return* std::sqrt(x) / (1 + 2\*x);}, 1, 2, i+1);

}

std::cout << "Integral 1:" << std::endl;

*for*(*const* *auto*& res : res1)

std::cout << res << std::endl;

std::cout << "\n\n Integral 2:" << std::endl;

*for*(*const* *auto*& res : res2)

std::cout << res << std::endl;

}

## Funktionszeiger Complex struct

#include <iostream>

*using* std::endl;

*using* std::cout;

*struct* **Complex**{

double re, im;

void **printMe**() *const*

{

cout << "(" << re << ","

<< im << ")" << endl;

}

};

Complex **plus42Version1**(Complex z) {

Complex erg = {z};

erg.re += 42;

erg.im += 42;

*return* erg;

}

*//Complex&,* *das* *&* *muss* *weg* *damit* *funktioniert*

Complex& **plus42Version2**(*const* Complex& z)

{

Complex erg = {z};

erg.re += 42;

erg.im += 42;

*return* erg;

}

Complex\* **plus42Version3**(*const* Complex& z) {

Complex erg = {z};

erg.re += 42;

erg.im += 42;

*return* &erg;

}

Complex\* **plus42Version4**(*const* Complex& z) {

Complex\* erg = *new* Complex{z};

erg->re += 42;

erg->im += 42;

*return* erg;

}

void **plus42Version5**(*const* Complex& z, Complex& erg) {

erg.re = z.re + 42;

erg.im = z.im + 42;

}

void **plus42Version6**(*const* Complex& z, Complex\* erg) {

erg->re = z.re + 42;

erg->im = z.im + 42;

}

int **seite114\_118**() {

Complex z1{1.0, 2.0};

Complex erg1 = plus42Version1(z1);

erg1.printMe();

Complex z2{1.0, 2.0};

Complex erg2 = plus42Version2(z2);

erg2.printMe();

Complex z3{1.0, 2.0};

Complex\* erg3 = plus42Version3(z3);

erg3->printMe();

Complex z4{1.0, 2.0};

Complex\* erg4 = plus42Version4(z4);

erg4->printMe();

*//meckert* *der* *Google* *Style-Checker* *cpplint* *an*

*//erg5* *durch* *die* *Funktion* *ver¨andert* *wird.*

Complex z5{1.0, 2.0};

Complex erg5{0, 0};

plus42Version5(z5, *erg5*);

erg5.printMe();

*//cpplint* *konform*

*//schreibenden* *Zugriff* *auf* *erg6* *hat*

Complex z6{1.0, 2.0};

Complex erg6{0, 0};

plus42Version6(z6, &erg6);

erg6.printMe();

*return* 0;

}

## Default Konstruktor

|  |  |
| --- | --- |
| *class* **Datum** {  *private*:  *static* *const* int jahrDefault = 1970,  monatDefault = 1,  tagDefault = 1;  *//* *Attribute*  int jahr, monat, tag;  *public*:  *//* *Vom* *Programmierer* *erstellter*  *//* *Default-Konstruktor*  **Datum**()  {  jahr = jahrDefault;  monat = monatDefault;  tag = tagDefault;  }  }; | Datum heute;  heute.printDatum();  Datum\* pd = *new* Datum;  pd->printDatum(); |

## Klasse Uhr

#include <iostream>

#include <iomanip>

*class* **Uhr**

{

*private*:

int stunden;

int minuten;

int sekunden;

*public*:

**Uhr**(int h, int m, int s) : stunden(h), minuten(m), sekunden(s) {}

**Uhr**() : stunden(0), minuten(0), sekunden(0) {}

*private*:

void **checkAttributes**()

{

*if* (stunden < 0 || stunden > 24)

{

stunden = 0;

}

*if* (minuten < 0 || minuten > 60)

{

minuten = 0;

}

*if* (sekunden < 0 || sekunden > 61) *//* *Schaltsekunde*

{

sekunden = 0;

}

}

*public*:

void **setTime**(int h, int m, int s)

{

stunden = h;

minuten = m;

sekunden = s;

checkAttributes();

}

void **tick**()

{

*if* (sekunden == 59)

{

*if* (minuten == 59)

{

*if* (stunden == 23)

{

stunden = 0;

}

*else*

{

++stunden;

}

minuten = 0;

}

*else*

{

++minuten;

}

sekunden = 0;

}

*else*

{

++sekunden;

}

}

void **displayClock**()

{

std::cout << std::setw(2) << std::setfill('0');

std::cout << stunden << ":";

std::cout << std::setw(2) << std::setfill('0');

std::cout << minuten << ":";

std::cout << std::setw(2) << std::setfill('0');

std::cout << sekunden << std::endl;

}

*virtual* ~***Uhr***()

{

std::cout << "Objekt stirbt" << std::endl;

}

};

void **seite198**()

{

Uhr u {};

*for* (int i = 0; i < 7200; ++i)

{

u.displayClock();

u.tick();

}

}

## Einfach Verkettete Liste

#include <cmath>

#include <iostream>

*struct* **ListenElement**

{

double wert;

ListenElement\* next;

};

*class* **Liste**

{

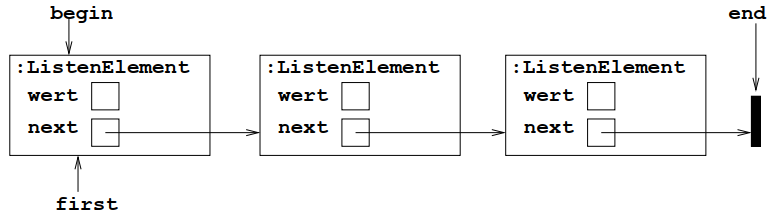
*private*:

int counter;

ListenElement\* first;

*public*:

**Liste**() : counter(0), first(*nullptr*) {}

 int **size**() *const*

{

*return* counter;

}

double **at**(int post) *const*

{

ListenElement\* pos = first;

int i = 0;

*while* (pos && i < post)

{

pos = pos->next;

++i;

}

*return* pos->wert;

}

double **erase\_front**() {

ListenElement\* first\_element = first;

double ret\_val = NAN;

*if* (first)

{

*if* (first->next)

{

first = first->next;

}

*else*

{

first = end();

}

ret\_val = first\_element->wert;

}

*delete* first\_element;

first\_element = *nullptr*;

*return* ret\_val;

--counter;

}

double **erase\_back**() {

double ret\_val = NAN;

ListenElement\* pos = first;

*if* (pos)

{

*while* (pos->next)

{

pos = pos->next;

}

ListenElement\* pos2 = first;

*while* (pos2->next != pos)

{

pos2 = pos2->next;

}

pos2->next = end();

ret\_val = pos->wert;

*delete* pos;

--counter;

}

*return* ret\_val;

}

void **push\_front**(double wert) {

ListenElement\* second = first;

first = *new* ListenElement;

first->wert = wert;

first->next = second;

++counter;

}

void **push\_back**(double wert) {

ListenElement\* to\_add = *new* ListenElement;

to\_add->wert = wert;

to\_add->next = end();

ListenElement\* pos = first;

*if* (pos)

{

*while* (pos->next)

{

pos = pos->next;

}

pos->next = to\_add;

}

*else*

{

first = to\_add;

}

++counter;

}

void **printList**() *const* {

ListenElement\* pos = begin();

int counter = 0;

*if*(pos) {

*while*(pos->next) {

std::cout << counter << ": " << pos->wert << std::endl;

++counter;

pos = pos->next;

}

}

}

*virtual* ~***Liste***() {

ListenElement\* pos = first;

*if*(pos) {

*while*(pos->next) {

ListenElement\* temp = pos->next;

*delete* pos;

pos = temp;

}

}

}

ListenElement\* **begin**() *const*

{

*return* first;

}

ListenElement\* **end**() *const*

{

*return* *nullptr*;

}

};

void **seite199**() {

Liste l {};

l.push\_front(1.0);

l.push\_front(2.0);

l.push\_front(3.0);

l.printList();

}

#endif *//* *SEITE199\_H*

## Operatoren + Rule of Three

#include <stdexcept>

#include <iostream>

*class* **Vec** {

*private*:

int n;

double\* elem;

*public*:

**Vec**() { *//* *Standard-Konstruktor*

n = 5;

elem = *new* double[n] {};

}

*explicit* **Vec**(int size) { *//* *nicht-konvertierend*

n = size;

elem = *new* double[n] {};

}

**Vec**(double wert) { *//* *konvertierend*

n=5;

elem = *new* double[n];

*for*(int i = 0; i<n; ++i) {

elem[i] = wert;

}

}

**Vec**(int size, double wert) : Vec(size) { *//* *delegierend*

*for*(int i = 0; i < n; ++i) {

elem[i] = wert;

}

}

double& *operator*[](int idx) { *//* *assignable* *-->* *reference*

*if*(idx < n && idx >= 0) {

*return* elem[idx];

} *else* {

*throw* std::out\_of\_range("Index must be positive and smaller than n!");

}

}

double *operator*[](int idx) *const* { *//* *const*

*if*(idx < n && idx >= 0) {

*return* elem[idx];

} *else* {

*throw* std::out\_of\_range("Index must be positive and smaller than n!");

}

}

Vec& *operator*+=(*const* Vec& v) { *//* *Vector* *+* *Vector*

*if*(v.n != *this*->n) {

*throw* std::domain\_error("Vectors must have same size!");

}

*for*(int i = 0; i < n; ++i) {

elem[i] += (v.elem)[i];

}

*return* \**this*;

}

Vec& *operator*+=(double wert) { *//* *Vektor* *+* *Zahl*

*for*(int i = 0; i < n; ++i) {

elem[i] += wert;

}

*return* \**this*;

}

Vec *operator*+(double wert) *const* {

Vec ret(n);

*for*(int i=0; i < n; ++i) {

ret[i] = elem[i] + wert;

}

*return* ret;

}

*virtual* ~***Vec***() {

*delete*[] elem;

}

Vec& *operator*=(*const* Vec& rhs) { *//* *copy* *assignment* *operator*

*if*(*this* != &rhs) {

*delete*[] elem; n = rhs.n;

elem = *new* double[n];

*for*(int i = 0; i < n; ++i) {

elem[i] = rhs[i];

}

}

*return* \**this*;

}

**Vec**(*const* Vec& rhs) {

n = rhs.n;

elem = *new* double[n];

*for*(int i = 0; i < n; ++i) {

elem[i] = rhs[i];

}

}

void **output**(std::ostream& os) *const* {

*for*(int i = 0; i < n-1; ++i) {

os << elem[i] << ", ";

}

*if*(n>0) {

os << elem[n-1] << std::endl;

}

}

};

*inline* std::ostream& *operator*<<(std::ostream& os, *const* Vec& v) {

v.output(*os*);

*return* os;

}

void **seite227**() {

Vec v(4, 2.5);

*//* *std::cout* *<<* *v;*

v += 2;

std::cout << "V: " << v;

Vec v2;

v2 = v;

v2 += 1;

v2[0] = 1;

v2[3] \*= 2;

std::cout << "V2: " << v2;

Vec vsum = v;

vsum += v2;

std::cout << "V+V2= " << vsum;

*try* {

std::cout << v2[8] << std::endl;

} *catch* (*const* std::out\_of\_range& e) {

std::cout << "Wrong index!" << std::endl;

}

}

## Klasse: Vater Figur, Kind Kreis, Kind Rechteck

#include <cmath>

#include <cstdlib>

#include <ctime>

#include <iostream>

*using* std::cout;

*using* std::endl;

*class* **Figur**

{

*private*:

double x;

double y;

*public*:

**Figur**(double x\_, double y\_) : x(x\_), y(y\_) {}

*virtual* void ***output***(std::ostream& os) *const*

{

os << "(" << x << ", " << y << ")";

}

*virtual* double ***flaeche***() *const* = 0;

};

*class* **Kreis** : *public* Figur

{

*private*:

double r;

*public*:

**Kreis**(double r\_, double x\_, double y\_) : Figur(x\_, y\_), r(r\_) {}

void ***output***(std::ostream& os) *const*

{

Figur::output(*os*);

os << ": Kreis mit r=" << r;

}

double ***flaeche***() *const*

{

*return* M\_PI \* r \* r;

}

};

*class* **Rechteck** : *public* Figur

{

*private*:

double a, b;

*public*:

**Rechteck**(double a\_, double b\_, double x\_, double y\_) : Figur(x\_, y\_), a(a\_), b(b\_) {}

*virtual* void ***output***(std::ostream& os) *const*

{

Figur::output(*os*);

os << ": Rechteck mit a=" << a << " und b=" << b;

}

*virtual* double ***flaeche***() *const*

{

*return* a \* b;

}

};

*inline* std::ostream& *operator*<<(std::ostream& os, *const* Figur& f)

{

f.*output*(*os*);

*//* *os* *<<* *std::endl;*

*return* os;

}

void **seite258**()

{

Rechteck r1(1, 2, 0, 0), r2(3, 4, 1, 2);

Kreis k1(1, 0, 0);

cout << "Rechtecke: " << std::endl;

cout << "r1: " << r1 << "; r2: " << r2 << std::endl;

cout << std::endl << "Kreise: " << std::endl;

cout << "k1: " << k1 << endl;

cout << "\nFlächen:" << endl;

cout << "r1: " << r1.*flaeche*() << "; r2: " << r2.*flaeche*() << endl;

cout << "k1: " << k1.*flaeche*() << endl;

}

#endif *//* *SEITE258\_H*

## Klasse ILebewesen Schnittstelle

#include <iostream>

*class* **ILebewesen** {

*public*:

*virtual* void ***gibLaut***() *const* = 0;

*virtual* double ***getLebensAlter***() *const* = 0;

*virtual* void ***output***(std::ostream& os) *const* = 0;

*virtual* ~***ILebewesen***() {}

};

*class* **Lebewesen** : *public* ILebewesen {

*private*:

int alter;

*public*:

**Lebewesen**(int a) : alter(a) {}

int **getAlter**() *const* {*return* alter;}

void ***output***(std::ostream& os) *const* {

os << "Lebewesen";

}

*virtual* ~***Lebewesen***() {}

};

*class* **Hund** : *public* Lebewesen {

*private*:

double gewicht;

*public*:

**Hund**(int a, double g) : Lebewesen(a), gewicht(g) {}

void ***output***(std::ostream& os) *const* {

Lebewesen::output(*os*);

os << " Hund, Lebensalter " << *getLebensAlter*() << " Jahre, Gewicht " << gewicht << "kg";

}

void ***gibLaut***() *const* {

std::cout << "Wau Wau" << std::endl;

}

double ***getLebensAlter***() *const* {

*return* getAlter() \* 5.5;

}

void **sitz**() {}

*virtual* ~***Hund***() {}

};

*class* **Wellensittich** : *public* Lebewesen {

*private*:

char geschlecht;

*public*:

**Wellensittich**(int a, char g) : Lebewesen(a), geschlecht(g) {}

void ***gibLaut***() *const* {

std::cout << "Piep Piep" << std::endl;

}

double ***getLebensAlter***() *const* {

*return* getAlter() \* 12.9;

}

void ***output***(std::ostream& os) *const* {

Lebewesen::output(*os*);

os << " Wellensittich, Lebensalter " << *getLebensAlter*() << " Jahre, Geschlecht " << geschlecht;

}

*virtual* ~***Wellensittich***() {}

};

*inline* std::ostream& *operator*<<(std::ostream& os, *const* ILebewesen& l) {

l.*output*(*os*);

*return* os;

}

void **seite263**() {

Hund h(5, 23.3);

Wellensittich w(2, 'd');

std::cout << h << std::endl;

std::cout << w << std::endl;

}

## Klasse Fahrzeug (zu schwer)

#include <iostream>

#include <string>

#include <sstream>

#include <cstdlib>

#include <list>

#include <algorithm>

#include <stdexcept>

*using* std::string;

*class* **Fahrzeug**

{

*private*:

int hub;

string kennzeichen;

*public*:

**Fahrzeug**(int h) : hub(h) {}

**Fahrzeug**() : Fahrzeug(0) {}

int **getHub**() *const*

{

*return* hub;

}

string **getKennzeichen**() *const*

{

*return* kennzeichen;

}

*virtual* double ***getKfzSteuer***() *const*

{

*return* (hub / 100.0) \* 5.0;

}

void **setKennzeichen**(string k)

{

kennzeichen = k;

}

*virtual* void ***output***(std::ostream& os) *const*

{

os << "Kennzeichen: " << kennzeichen << ", "

<< "Hub: " << hub << std::endl;

os << "KFZ-Steuer: " << *getKfzSteuer*() << std::endl;

}

};

*class* **Saisonfahrzeug** : *public* Fahrzeug

{

int startMonat, endeMonat;

*public*:

**Saisonfahrzeug**(int h, int sM, int eM)

: Fahrzeug(h), startMonat(sM), endeMonat(eM) {}

**Saisonfahrzeug**()

: Saisonfahrzeug(0, 0, 0) {}

double ***getKfzSteuer***() *const*

{

*return* (getHub() / 200.0) \* 1 \* (endeMonat - startMonat + 1);

}

void ***output***(std::ostream& os) *const*

{

Fahrzeug::output(*os*);

os << "Start-Monat: " << startMonat <<

", Ende-Monat: " << endeMonat << std::endl;

}

};

*class* **Nutzfahrzeug** : *public* Fahrzeug

{

int jahresKm;

*public*:

**Nutzfahrzeug**(int h, int jkm)

: Fahrzeug(h), jahresKm(jkm) {}

double ***getKfzSteuer***() *const*

{

*return* jahresKm \* 0.02 \* getHub() / 10000.0;

}

void ***output***(std::ostream& os) *const*

{

Fahrzeug::output(*os*);

os << "Jahres-km: " << jahresKm << std::endl;

}

};

*class* **Halter**

{

*private*:

string name;

std::list<Fahrzeug\*> fahrzeugListe;

*public*:

**Halter**(string n)

: name(n), fahrzeugListe() {}

void **fahrzeugAufnehmen**(Fahrzeug\* f)

{

fahrzeugListe.push\_back(f);

}

void **fahrzeugLoeschen**(Fahrzeug\* f)

{

fahrzeugListe.remove(f);

}

double **getKFZSteuer**() *const*

{

double steuer = 0.0;

*for* (*const* Fahrzeug \* *const*& f : fahrzeugListe)

{

steuer += f->*getKfzSteuer*();

}

*return* steuer;

}

int **anzahlFahrzeuge**() *const*

{

*return* fahrzeugListe.size();

}

void **output**(std::ostream& os) *const*

{

os << "Name: " << name << std::endl;

*for* (*const* Fahrzeug \* *const*& f : fahrzeugListe)

{

os << "--------------------------------" << std::endl;

f->*output*(*os*);

}

os << "Gesamt-KFZ-Steuer: " << getKFZSteuer() << std::endl;

}

};

*class* **KennzeichenGenerator**

{

*static* *const* int anfangsZahl = 500;

*static* *const* int endZahl = 9999;

*static* *const* char anfangsBuchstabe = 'A';

*static* *const* char endBuchstabe = 'Z';

string ort;

*public*:

**KennzeichenGenerator**(string o)

: ort(o) {}

string **getNext**() *const*

{

std::stringstream kz;

kz << ort << " ";

*for* (int i = 0; i < 2; ++i)

{

kz << *static\_cast*<char>(anfangsBuchstabe + rand() % (endBuchstabe - anfangsBuchstabe));

}

kz << " ";

int rand\_num = *static\_cast*<double>(rand()) / RAND\_MAX \* (endZahl - anfangsZahl) + anfangsZahl;

kz << rand\_num;

*return* kz.str();

}

};

*class* **Zulassungsstelle** {

*private*:

string landkreisName;

string ortsKuerzel;

std::list<Halter\*> halterListe;

*const* KennzeichenGenerator kg;

*public*:

**Zulassungsstelle**(string ln, string ok)

: landkreisName(ln), ortsKuerzel(ok), kg(ok) {}

string **anmelden**(Halter\* h, Fahrzeug\* f) {

*//* *Kennzeichen* *generieren*

string kennzeichen = kg.getNext();

f->setKennzeichen(kennzeichen);

*//* *check* *whether* *Halter* *is* *already* *registered*

*const* *auto* halter\_it = std::find(halterListe.begin(), halterListe.end(), h);

*if*(halter\_it != halterListe.end()) {

*//* *Halter* *already* *exists* *-->* *append* *Fahrzeug*

(\*halter\_it)->fahrzeugAufnehmen(f);

} *else* {

*//* *append* *Fahrzeug* *to* *Halter* *and* *add* *Halter* *to* *list*

h->fahrzeugAufnehmen(f);

halterListe.push\_back(h);

}

*return* kennzeichen;

}

void **abmelden**(Halter\* h, Fahrzeug\* f) {

*const* *auto* halter\_it = std::find(halterListe.begin(), halterListe.end(), h);

*if*(halter\_it == halterListe.end()) {

*throw* std::out\_of\_range("Halter nicht im System!");

}

(\*halter\_it)->fahrzeugLoeschen(f);

}

double **getKFZSteuer**() *const* {

double steuer = 0.0;

*for*(*const* *auto*& h : halterListe) {

steuer += h->getKFZSteuer();

}

*return* steuer;

}

void **output**(std::ostream& os) *const* {

os << "Zulassungsstelle im Landkreis: " << landkreisName << std::endl;

*for*(*const* *auto*& h : halterListe) {

os << "=========================================" << std::endl;

h->output(*os*);

}

os << "Gesamtes KFZ-Steuer Aufkommen: " << getKFZSteuer() << std::endl;

}

};

void **seite268**()

{

Zulassungsstelle zs("Musterland", "Mu");

Halter h1("Mustermann");

Fahrzeug f1(3000);

zs.anmelden(&h1, &f1);

Fahrzeug f2(5000);

zs.anmelden(&h1, &f2);

Saisonfahrzeug sf1(6000, 3, 10);

zs.anmelden(&h1, &sf1);

Halter h2("BauerXaver");

Fahrzeug f3(4000);

zs.anmelden(&h2, &f3);

Nutzfahrzeug nf1(7000, 10000);

zs.anmelden(&h2, &nf1);

zs.output(std*::cout*);

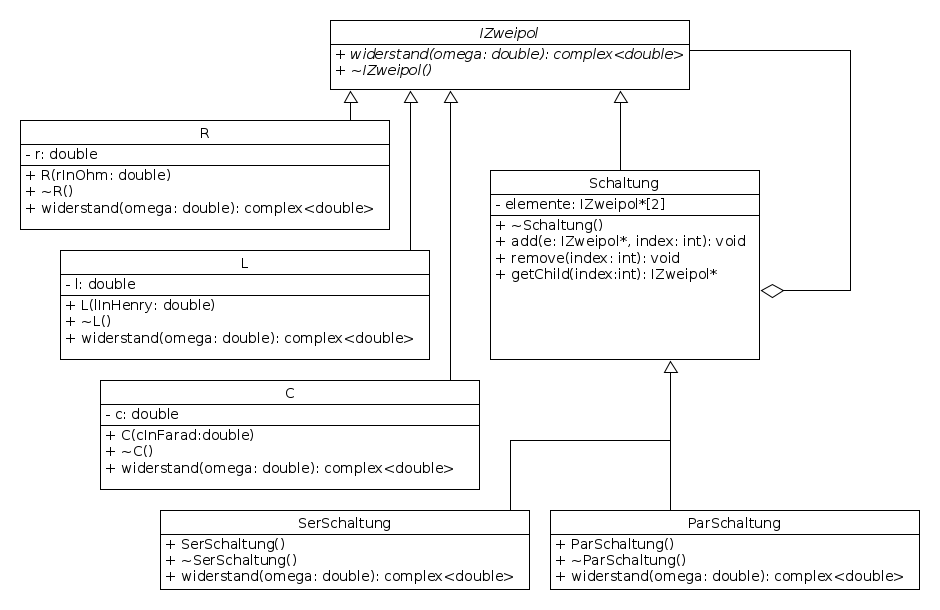
zs.abmelden(&h1, &f2);

zs.abmelden(&h2, &f3);

zs.output(std*::cout*);

}

## IZweipol (schaltung RLC)

#ifndef SEITE281\_H

#define SEITE281\_H

#include <complex>

#include <iostream>

#include <cmath>

*using* std::complex;

*class* **IZweipol** {

*public*:

*virtual* complex<double> ***widerstand***(double omega) *const* = 0;

*virtual* ~***IZweipol***() {}

};

*class* **R** : *public* IZweipol {

double r;

*public*:

**R**(double rInOhm) : r(rInOhm) {}

*virtual* ~***R***() {}

complex<double> ***widerstand***(*\_\_attribute\_\_*((unused)) double omega) *const* *override* {

*return* r;

}

};

*class* **L** : *public* IZweipol {

double l;

*public*:

**L**(double lInHenry) : l(lInHenry) {}

*virtual* ~***L***() {}

complex<double> ***widerstand***(double omega) *const* *override* {

*return* complex<double>(0.0, omega \* l);

}

};

*class* **C** : *public* IZweipol {

double c;

*public*:

**C**(double cInFarad) : c(cInFarad) {}

*virtual* ~***C***() {}

complex<double> ***widerstand***(double omega) *const* *override* {

*return* complex<double>(0.0, -1.0 / (omega \* c));

}

};

*class* **Schaltung** : *public* IZweipol {

IZweipol\* elemente[2];

*public*:

**Schaltung**(IZweipol\* z1=*nullptr*, IZweipol\* z2=*nullptr*) : elemente{z1, z2} {}

*virtual* ~***Schaltung***() {}

void **add**(IZweipol\* e, int index) {

elemente[index] = e;

}

void **remove**(int index) {

elemente[index] = *nullptr*;

}

IZweipol\* **getChild**(int index) *const* {

*return* elemente[index];

}

};

*class* **SerSchaltung** : *public* Schaltung {

*public*:

*using* Schaltung::Schaltung;

*virtual* ~***SerSchaltung***() {}

complex<double> ***widerstand***(double omega) *const* *override* {

*//* *output* *stays* *infinity* *until* *both* *IZweipole* *are* *connected*

complex<double> retval = INFINITY;

*if*(getChild(0) && getChild(1)) {

retval = getChild(0)->*widerstand*(omega) +

getChild(1)->*widerstand*(omega);

}

*return* retval;

}

};

*class* **ParSchaltung** : *public* Schaltung {

*public*:

*using* Schaltung::Schaltung;

*virtual* ~***ParSchaltung***() {}

complex<double> ***widerstand***(double omega) *const* *override* {

complex<double> retval = INFINITY;

complex<double> w1=INFINITY, w2=INFINITY;

*if*(getChild(0)) {

w1 = getChild(0)->*widerstand*(omega);

}

*if*(getChild(1)) {

w2 = getChild(1)->*widerstand*(omega);

}

retval = 1.0/(1.0/w1 + 1.0/w2);

*return* retval;

}

};

void **seite281**() {

R r1(5), r2(10), r3(10);

L l1(1e-3), l2(2e-3);

C c1(1e-6);

SerSchaltung s1(&r3, &l2), s2(&r2, &c1), s3(&r1, &l1);

ParSchaltung p1(&s1, &s2);

SerSchaltung s4(&s3, &p1);

*for*(double d = 0; d < 8; d+=0.5) {

double omega = std::pow(10, d);

std::cout << "w=10^" << d << ": "

<< std::abs(s4.*widerstand*(omega)) << " Ohm" << std::endl;

}

}

## Observer

#include <iostream>

#include <list>

*using* std::cout;

*using* std::endl;

*class* **Observer** {

*public*:

*virtual* void ***update***() = 0;

};

*class* **Publisher** {

*private*:

std::list<Observer\*> observers;

*public*:

void **attatch**(Observer\* obs) {observers.push\_back(obs);}

void **detatch**(Observer\* obs) {observers.remove(obs);}

void **notify**() *const* {*for* (*auto* o : observers) {o->*update*();}

}

};

*class* **TemperaturSensor** : *public* Publisher {

*private*:

double temperatur;

*public*:

void **setTemperatur**(double t)

{

temperatur = t;

notify();

}

double **getTemperatur**() *const* { *return* temperatur; }

};

*class* **DezimalAnzeige** : *public* Observer {

*private*:

double temperatur;

TemperaturSensor\* publisher;

*public*:

**DezimalAnzeige**(TemperaturSensor\* pub)

: temperatur(pub->getTemperatur()),publisher(pub)

{

publisher->attatch(*this*);

}

void **display**() *const*

{

cout << "Temperatur: "

<< temperatur << endl;

}

void ***update***() *override*

{

temperatur = publisher->getTemperatur();

display();

}

~**DezimalAnzeige**() { publisher->detatch(*this*); }

};

*class* **LEDSignalAnzeige** : *public* Observer {

*private*:

*static* int greenMin;

*static* int greenMax;

*enum* *class* **MyColor** { *green*, *red* };

MyColor color;

TemperaturSensor\* publisher;

*public*:

**LEDSignalAnzeige**(TemperaturSensor\* pub)

: color(MyColor::*green*), publisher(pub)

{

publisher->attatch(*this*);

}

void **display**() *const*

{

cout << "LED-Anzeige: "

<< ((color == MyColor::*green*) ? "green" : "red")

<< endl;

}

void ***update***() *override*

{

double t = publisher->getTemperatur();

*if* (greenMin <= t && t <= greenMax) {

color = MyColor::*green*;

} *else* {

color = MyColor::*red*;

}

display();

}

~**LEDSignalAnzeige**() { publisher->detatch(*this*); }

};

int LEDSignalAnzeige::greenMin = 16;

int LEDSignalAnzeige::greenMax = 25;

int **main1**() {

TemperaturSensor ts;

DezimalAnzeige da{&ts};

LEDSignalAnzeige lsa{&ts};

ts.setTemperatur(22);

ts.setTemperatur(15.5);

ts.detatch(&da);

ts.setTemperatur(21.3);

ts.setTemperatur(25.4);

*return* 0;

}

*/\** *Ausgabe*

*Temperatur:* *22*

*LED-Anzeige:* *green*

*Temperatur:* *15.5*

*LED-Anzeige:* *red*

*LED-Anzeige:* *green*

*LED-Anzeige:* *red*

*\*/*

## Bubble Sort

void **swap**(int& x, int& y) {

int temp = x;

x = y;

y = temp;

}

bool **less**(int x, int y) { *return* x < y; }

void **bubbleSort**(int\* v, int n) {

*for* (int i = 1; i < n; ++i) {

*for* (int j = 0; j < n - i; ++j) {

*if* (less(v[j + 1], v[j])) {

swap(*v*[*j*], *v*[*j* + 1]);

}

}

}

}

void **printVec**(int\* v, int n) {

*for* (int i = 0; i < n; ++i) {

cout << v[i] << endl;

}

}

int main() {

int v[5] = {10, 8, 3, 2, 7};

bubbleSort(v, 5);

printVec(v, 5);

*return* 0;

}

## Template Stack

#include <stdexcept>

#include <iostream>

#include <vector>

#include <cstring>

*template* <*class* T>

*class* **Stack**

{

*private*:

int size;

int tos;

T\* data;

*public*:

**Stack**(int s) : size(s), tos(-1), data(*new* T[s]) {}

void **push**(T element)

{

*if* (tos >= size - 1)

{

*throw* std::out\_of\_range("Stack overflow!");

}

*else*

{

++tos;

data[tos] = element;

}

}

bool **empty**() *const*

{

*return* tos == -1;

}

T **pop**()

{

*if* (tos < 0)

{

*throw* std::out\_of\_range("Stack underflow!");

}

*else*

{

--tos;

*return* data[tos + 1];

}

}

T **top**() *const*

{

*if* (tos < 0)

{

*throw* std::out\_of\_range("Stack underflow!");

}

*else*

{

*return* data[tos];

}

}

void **output**() *const*

{

*for* (int i = tos; i >= 0; --i)

{

std::cout << data[i] << " ";

}

std::cout << std::endl;

}

*virtual* ~***Stack***()

{

*delete*[] data;

}

};

*inline* bool **isNumeric**(char c)

{

*return* c >= '0' && c <= '9';

}

*inline* bool **isOperator**(char c)

{

*return* c == '+' || c == '-' || c == '\*' || c == '/';

}

void **infixToPostfix**(*const* char\* infix, char\* postfix)

{

int len = strlen(infix);

Stack<char> s(len);

int k = 0; *//* *position* *in* *output* *string*

*for* (int i = 0; i < len; ++i)

{

char c = infix[i];

*if* (isOperator(c))

{

s.push(c); *//* *push* *operator* *on* *stack*

}

*if* (isNumeric(c))

{

postfix[k++] = c;

*//* *read* *until* *end* *of* *number*

*while* (isNumeric(infix[i + 1]))

{

postfix[k++] = infix[++i];

}

*//* *add* *space* *to* *separate*

postfix[k++] = ' ';

}

*//* *assign* *letter* *again* *in* *case* *i* *has* *changed*

c = infix[i];

*if* (c == ')')

{

postfix[k++] = s.pop();

postfix[k++] = ' ';

}

}

postfix[k] = '\0';

}

double **postfixToValue**(*const* char\* postfix)

{

int len = strlen(postfix);

Stack<double> s(len);

char\* *const* copy = *new* char[len+1];

strcpy(copy, postfix);

*//* *create* *movable* *pointer*

char\* pc = copy;

*while* (\*pc != '\0')

{

*if* (isNumeric(\*pc))

{

*//* *reads* *in* *double* *until* *blank* *space* *or* *operator*

*//* *and* *moves* *pc* *to* *the* *following* *character*

double val = strtod(pc, &pc);

s.push(val);

}

*if* (isOperator(\*pc))

{

double v1 = s.pop();

double v2 = s.pop();

double res;

*if* (\*pc == '+')

{

res = v2 + v1;

}

*else* *if* (\*pc == '-')

{

res = v2 - v1;

}

*else* *if* (\*pc == '\*')

{

res = v2 \* v1;

}

*else* *if* (\*pc == '/')

{

res = v2 / v1;

}

*else*

{

*throw* std::invalid\_argument("Invalid operator!");

}

s.push(res);

}

*//* *next* *character*

++pc;

}

*delete*[] copy;

*return* s.top();

}

void **seite336**()

{

Stack<int> s(10);

std::vector<int> vec = {3, 69, 42, 10, 420};

*for* (*auto* v : vec)

{

s.push(v);

}

s.output();

s.pop();

s.pop();

s.output();

*const* char\* infix = "(((15 - 3) \* 4) + (6 /(2 + 1)))";

char\* postfix = *new* char[strlen(infix) \* 2];

infixToPostfix(infix, postfix);

std::cout << "Infix: " << infix << std::endl;

std::cout << "Postfix: " << postfix << std::endl;

double val = postfixToValue(postfix);

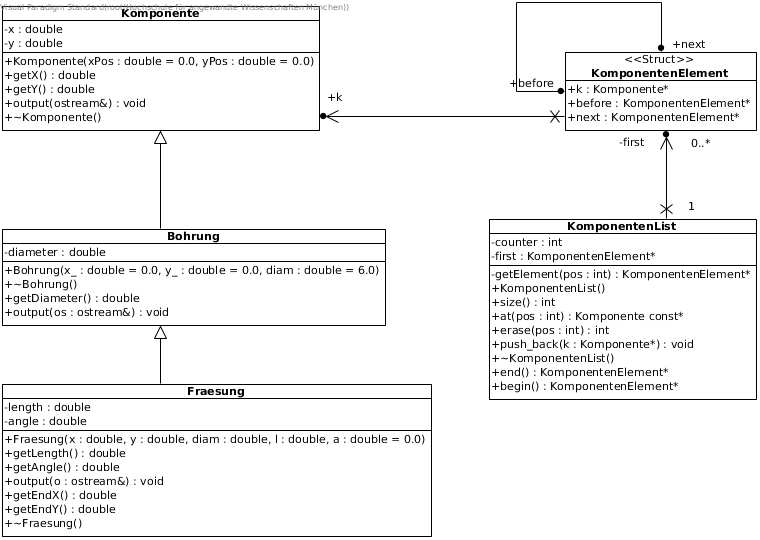
std::cout << "Value: " << val << std::endl;

*delete*[] postfix;

}

# Praktikum

## P1



### Komponente

#ifndef KOMPONENTE\_H

#define KOMPONENTE\_H

#include <iostream>

*class* **Komponente**

{

*//private:*

double x, y;

*public*:

*//Konstruktor*

**Komponente**(double xPos = 0.0, double yPos = 0.0)

: x(xPos), y(yPos) {}

*virtual* double ***getX***() *const* {*return* x;}

*virtual* double ***getY***() *const* {*return* y;}

*virtual* void ***output***(std::ostream& os) *const*

{

os << "(" << x << ", " << y << ")";

}

*virtual* ~***Komponente***() {}

};

*inline* std::ostream& *operator*<<(std::ostream& os, *const* Komponente& k)

{

k.*output*(*os*);

*return* os;

}

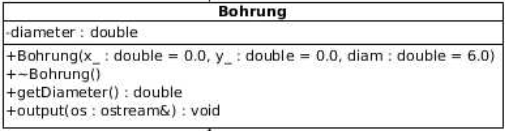
### Bohrung

#include "Komponente.h"

#include <cmath>

#include <iostream>

*using* std::endl;

*class* **Bohrung** : *public* Komponente

{

*//private:*

double diameter;

*public*:

**Bohrung**(double x\_ = 0.0, double y\_= 0.0, double diam = 6.0)

: Komponente(x\_, y\_), diameter(diam)*//diameter(diam* *>* *0* *?* *diam* *:* *6.0){}*

{

*if* (diameter > 0.0){diameter = diam;}

*else* {diameter = 6.0;}

}

double **getDiameter**() *const* {*return* diameter;}

void ***output***(std::ostream &os) *const*

{

os << "Bohrung: ";

Komponente::output(*os*);

os << ", Durchmesser: " << diameter;

}

~***Bohrung*** () {}

};

### Fraesung

#include <Bohrung.h>

#include <cmath>

#include <iostream>

*using* std::endl;

*class* **Fraesung** : *public* Bohrung

{

*//private:*

double length;

double angle;

*public*:

**Fraesung**(double x, double y, double diam, double l, double a = 0.0)

:Bohrung(x, y, diam), length(l), angle(a) */\*length(l>0* *?* *l* *:* *0.0)\*/*

{

*if*(length > 0){length = l;}

*else*{length = 0;}

}

double **getLength**() *const* {*return* length;}

double **getAngle**() *const* {*return* angle;}

void ***output***(std::ostream &os) *const*

{

os << "Fraesung mit Start: ";

Komponente::output(*os*);

os << " und Endpunkt: (" << getEndX() << ", " << getEndY() << ")"

<< ", Durchmesser: " << getDiameter();

}

double **getEndX**() *const*{*return* *getX*() + length \* std::cos(angle);}

double **getEndY**() *const*{*return* *getY*() + length \* std::sin(angle);}

~***Fraesung***() {}

};

### KomponentenElement

#include <Komponente.h>

*struct* **KomponentenElement**

{

Komponente\* k;

KomponentenElement\* before;

KomponentenElement\* next;

};

### KomponentenListe

#include "KomponentenElement.h"

#include "Komponente.h"

*class* **KomponentenList**

{

*//private:*

int counter;

KomponentenElement\* first;

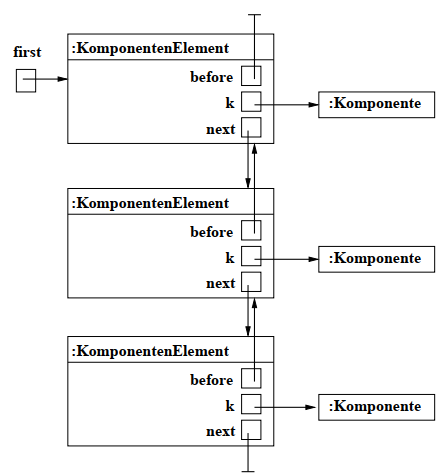
KomponentenElement\* **getElement**(int pos) *const*;

*public*:

**KomponentenList**()

:counter(0), first(*nullptr*){}

int **size**() *const*

 {

*return* counter;

}

int **erase**(int pos);

void **push\_back**(Komponente\* k);

KomponentenElement\* **begin**() *const*

{

*return* first;

}

KomponentenElement\* **end**() *const*

{

*return* *nullptr*;

}

Komponente *const*\* **at**(int pos) *const*;

~**KomponentenList**();

};

*///////////////////////////////////////////////////////////////////////////////////////*

#include "KomponentenList.h"

#include "Komponente.h"

*///////////////////////////////////////////////////////////////////////////////////////*

KomponentenElement\* KomponentenList::**getElement**(int pos) *const*

{

KomponentenElement\* tmp = *nullptr*;

*if*((pos>=0) && (pos < counter))

{

tmp = first;

*//laufen* *bis* *pos*

*for*(int i = 0; i < pos; ++i)

{

tmp = tmp->next;

}

*return* tmp;

}

*else*

{

*return* *nullptr*;

}

}

*///////////////////////////////////////////////////////////////////////////////////////*

*const* Komponente\* KomponentenList::**at**(int pos) *const*

{

*//Gibt* *im* *tmp* *die* *position*

KomponentenElement\* tmp = getElement(pos);

*if*(pos>=0 && pos<counter)

{

*//ausgabe* *von* *komponent*

*return* tmp->k;

}

*return* *nullptr*;

}

*///////////////////////////////////////////////////////////////////////////////////////*

int KomponentenList::**erase**(int pos)

{

KomponentenElement\* tmp = getElement(pos);

KomponentenElement\* vorg = *nullptr*;

KomponentenElement\* nachf = *nullptr*;

*if*(counter == 0)

{

*return* -1;

}

*//solange* *position* *!=* *nullptr*

*if*(tmp != *nullptr*)

{

*//solange* *position* *nicht* *am* *Anfang*

*if*(tmp != first)

{

*//vorg* *=* *"tmp[n-1]"*

vorg = tmp->before;

*//nachf* *=* *"tmp[n+1]*

nachf = tmp->next;

*//solange* *nachf* *!=* *nullptr*

*if*(nachf != *nullptr*)

{

*//nachf[n-1]* *=* *tmp[n-1]*

nachf->before = tmp->before;

}

*//solange* *vorg* *!=* *nullptr*

*if*(vorg != *nullptr*)

{

*//vorg[n+1]* *=* *tmp[n+1]*

vorg->next = tmp->next;

}

}

*//sobald* *tmp* *==* *first*

*else*

{

*//erster* *ind.* *vom* *tmp[0+1]*

first = tmp->next;

}

*delete* tmp;

--counter;

*return* pos;

}

*return* -1;

}

*///////////////////////////////////////////////////////////////////////////////////////*

void KomponentenList::**push\_back**(Komponente \*k)

{

KomponentenElement\* tmp = *new* KomponentenElement();

tmp->k = k;

tmp->before = *nullptr*;

tmp->next = *nullptr*;

*if*(first == *nullptr*)

{

first = tmp;

}

*else*

{

KomponentenElement\* last = first;

*while*(last->next!=*nullptr*)

{

last = last->next;

}

last->next = tmp;

tmp->before=last;

}

++counter;

}

*///////////////////////////////////////////////////////////////////////////////////////*

KomponentenList::~**KomponentenList**()

{

KomponentenElement\* element\_to\_delete = first;

*while*(element\_to\_delete)

{

KomponentenElement\* next\_element = element\_to\_delete->next;

*delete* element\_to\_delete;

element\_to\_delete = next\_element;

}

}

### Main

#include <iostream>

#include "Bohrung.h"

#include "Komponente.h"

#include "Fraesung.h"

#include "KomponentenElement.h"

#include "KomponentenList.h"

#include <cmath>

*using* *namespace* std;

*using* std::endl;

*using* std::cout;

double **calcLength**(double\* co)

{

double dx = co[2] - co[0];

double dy = co[3] - co[1];

*return* sqrt(dx \* dx + dy \* dy);

}

double **calcAngle**(double \*co)

{

*return* atan2(co[2] - co[0], co[3] - co[1]);

}

int **main**()

{

Bohrung b1(1, 2, 8);

Bohrung b2(10, 8, 7);

Bohrung b3(7, 2, 5);

Bohrung b4(2, 6, 4);

*//double* *f1co[]* *=* *{2,* *3,* *4,* *5};*

*//double* *f2co[]* *=* *{6,* *7,* *9,* *7};*

*//double* *f3co[]* *=* *{17,* *6,* *13,* *2};*

Fraesung f1(2.0,3.0, 5.0, 2\*M\_SQRT2 , M\_PI\_4);

Fraesung f2(6.0, 7.0, 6.0, 3, 0);

Fraesung f3(17.0, 6.0, 4.0, (-4\*M\_SQRT2), M\_PI\_4);

cout << b1 << endl;

cout << f1 << endl;

cout << b3 << endl;

cout << f2 << endl;

cout << b4 << endl;

cout << f3 << endl;

cout << b2 << endl;

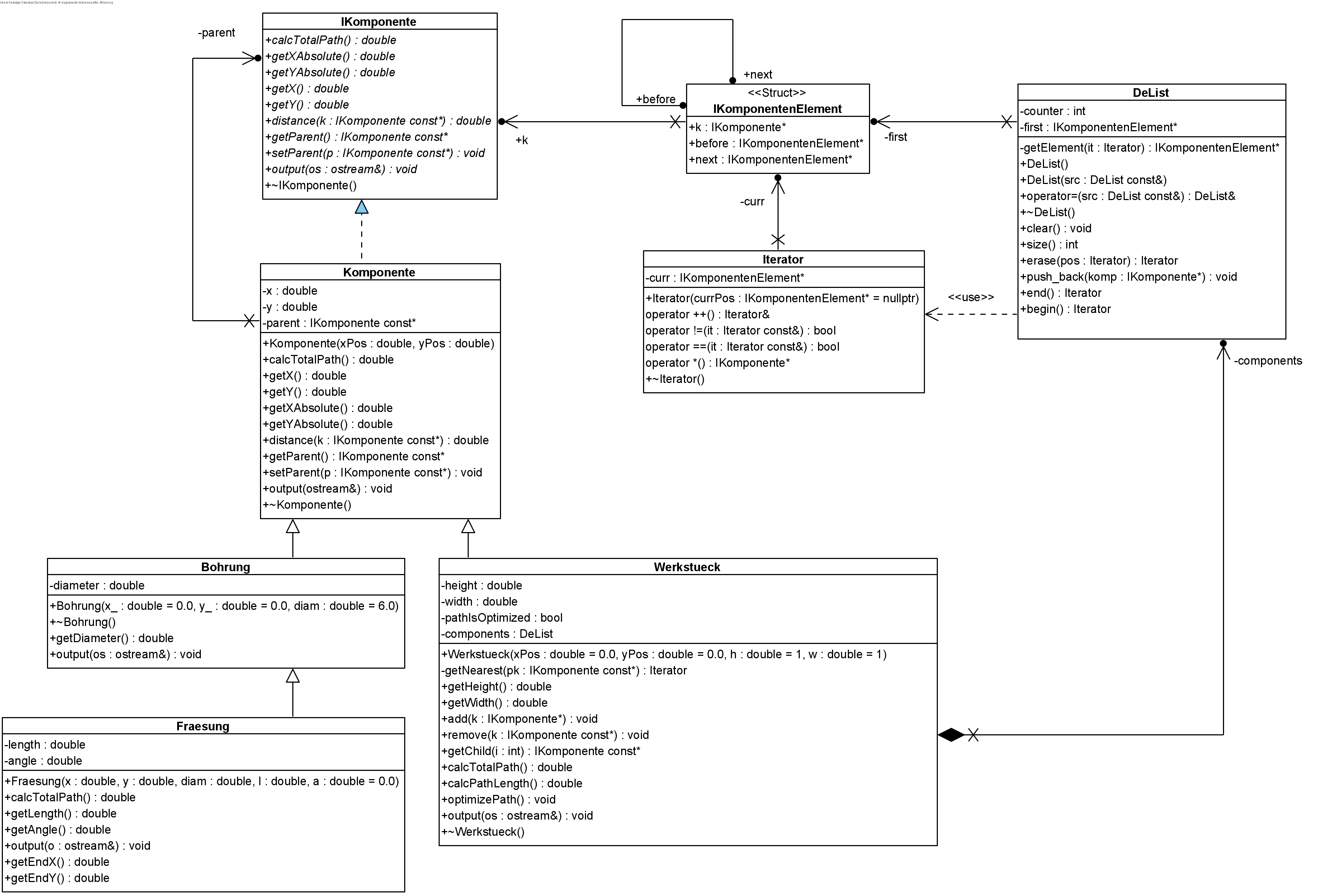
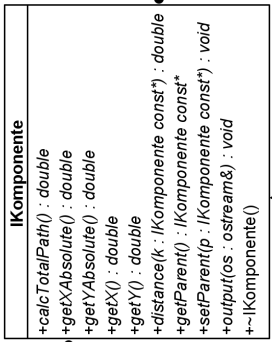
KomponentenList liste1{};

liste1.push\_back(&b1);

*return* 0;

}

## P2



### IKomponenten

#include <iostream>

*using* std::cout;

*using* std::endl;

*class* **IKomponente**

{

*public*:

*virtual* double ***calcTotalPath***() *const* = 0;

*virtual* double ***getXAbsolute***() *const* = 0;

*virtual* double ***getYAbsolute***() *const* = 0;

*virtual* double ***getX***() *const* = 0;

*virtual* double ***getY***() *const* = 0;

*virtual* double ***distance***(IKomponente *const* \*k) *const* = 0;

*virtual* IKomponente *const*\* ***getParent***() *const* = 0;

*virtual* void ***setParent***(IKomponente *const* \*p) = 0;

*virtual* void ***output***(std::ostream& os) *const* = 0;

*virtual* ~***IKomponente***();

};

*inline* std::ostream& *operator*<<(std::ostream& os, *const* IKomponente& ik)

{

ik.*output*(*os*);

*return* os;

}

IKomponente::~***IKomponente***() {};

### IKomponentenElement

#include "IKomponente.h"

*struct* **IKomponentenElement**

{

IKomponente\* k;

IKomponentenElement\* before;

IKomponentenElement\* next;

};

### Komponente

#include <iostream>

#include "IKomponente.h"

*class* **Komponente** : *public* IKomponente

{

*//private:*

double x, y;

IKomponente *const*\* parent;

*public*:

**Komponente**(double xPos = 0.0, double yPos = 0.0)

: x(xPos), y(yPos), parent(*nullptr*) {}

double ***calcTotalPath***() *const* *override*;

double ***getX***() *const* *override* {*return* x;}

double ***getY***() *const* *override* {*return* y;}

double ***getXAbsolute***() *const* *override*;

double ***getYAbsolute***() *const* *override*;

double ***distance***(IKomponente *const* \*k) *const* *override*;

IKomponente *const*\* ***getParent***() *const* *override*;

void ***setParent***(IKomponente *const* \*p) *override*;

*virtual* void ***output***(std::ostream& os) *const* *override*;

*virtual* ~***Komponente***() *override* {}

};

***////////////////////////////////////////////////////////////////////////////////***

#include "Komponente.h"

#include <cmath>

***////////////////////////////////////////////////////////////////////////////////***

void Komponente::***output***(std::ostream& os) *const*

{

os << "(" << x << ", " << y << ")" ;

}

***////////////////////////////////////////////////////////////////////////////////***

void Komponente::***setParent***(*const* IKomponente \*p)

{

parent = p;

}

***////////////////////////////////////////////////////////////////////////////////***

IKomponente *const*\* Komponente::***getParent***() *const*

{

*return* parent;

}

***////////////////////////////////////////////////////////////////////////////////***

double Komponente::***calcTotalPath***() *const*

{

*return* 0.0;

}

***////////////////////////////////////////////////////////////////////////////////***

double Komponente::***getXAbsolute***() *const*

{

double X = x;

*if*(parent != *nullptr*){

X = x + parent->*getXAbsolute*();

}

*return* X;

}

***////////////////////////////////////////////////////////////////////////////////***

double Komponente::***getYAbsolute***() *const*

{

double Y = y;

*if*(parent != *nullptr*)

{

Y = y + parent->*getYAbsolute*();

}

*return* Y;

}

***////////////////////////////////////////////////////////////////////////////////***

double Komponente::***distance***(*const* IKomponente \*k) *const*

{

double dX, dY;

dX = *getXAbsolute*() - k->*getXAbsolute*();

dY = *getYAbsolute*() - k->*getYAbsolute*();

*return* (sqrt(dX\*dX + dY\*dY));

}

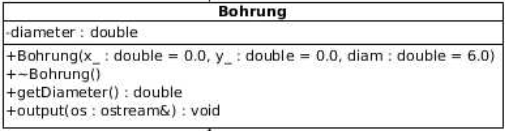
### Bohrung

#include "Komponente.h"

#include <cmath>

#include <iostream>

*using* std::endl;

*class* **Bohrung** : *public* Komponente

{

*//private:*

double diameter;

*public*:

**Bohrung**(double x\_ = 0.0, double y\_= 0.0, double diam = 6.0)

: Komponente(x\_, y\_), diameter(diam)*//diameter(diam* *>* *0* *?* *diam* *:* *6.0){}*

{

*if* (diameter > 0.0){diameter = diam;}

*else* {diameter = 6.0;}

}

double **getDiameter**() *const* {*return* diameter;}

void ***output***(std::ostream &os) *const*

{

os << "Bohrung: ";

Komponente::output(*os*);

os << ", Durchmesser: " << diameter;

}

~***Bohrung*** () {}

};

### Fraesung

#include <Bohrung.h>

#include <cmath>

#include <iostream>

*using* std::cout;

*using* std::endl;

*class* **Fraesung** : *public* Bohrung

{

double length;

double angle;

*public*:

**Fraesung**(double x, double y, double diam, double l, double a = 0.0)

:Bohrung(x, y, diam), length(l), angle(a) */\*length(l>0* *?* *l* *:* *0.0)\*/*

{

*if*(length > 0)

{

length = l;

}

*else*

{

length = 0;

}

}

double ***calcTotalPath***() *const* *override*;

double **getLength**() *const* {*return* length;}

double **getAngle**() *const* {*return* angle;}

void ***output***(std::ostream &os) *const* *override*

{

os << "Fraesung mit Start: ";

Komponente::output(*os*);

os << " und Endpunkt: (" << getEndX() << ", " << getEndY() << ")"

<< ", Durchmesser: " << getDiameter();

}

double **getEndX**() *const*;

double **getEndY**() *const*;

*virtual* ~***Fraesung***() *override* {}

};

#include "Fraesung.h"

***////////////////////////////////////////////////////////////////////////////////***

double Fraesung::**getEndX**() *const*

{

*return* (cos(*this*->getAngle()) \* *this*->getLength()) + *this*->*getX*();

}

***////////////////////////////////////////////////////////////////////////////////***

double Fraesung::**getEndY**() *const*

{

*return* (sin(*this*->getAngle())\* *this*->getLength()) + *this*->*getY*();

}

***////////////////////////////////////////////////////////////////////////////////***

double Fraesung::***calcTotalPath***() *const*

{

*return* 2 \* getLength();

}

### Iterator

#include "IKomponente.h"

#include "IKomponentenElement.h"

*class* **Iterator**

{

IKomponentenElement\* curr;

*public*:

**Iterator**(IKomponentenElement\* currPos = *nullptr*)

:curr(currPos) {}

Iterator& *operator*++()

{

*if*(curr != *nullptr*)

{

curr = curr->next;

}

*return* \**this*;

}

bool *operator*!=(Iterator *const*& it) *const*

{

*return* *this*->curr != it.curr;

}

bool *operator*==(Iterator *const*& it) *const*

{

*return* *this*->curr == it.curr;

}

IKomponente\* *operator*\*() *const*

{

IKomponente\* tmp = *nullptr*;

*if*(curr)

{

*//Rückgabe* *der* *aktuellen* *Elements* *(komponents)*

tmp = curr->k;

}

*return* tmp;

}

};

### DeList

#include "Komponente.h"

#include "IKomponentenElement.h"

#include "IKomponente.h"

#include "Iterator.h"

*class* **DeList**

{

int counter;

IKomponentenElement\* first;

IKomponentenElement\* **getElement**(Iterator it) *const*;

*public*:

**DeList**() : counter(0), first(*nullptr*) {}

*//copy* *Konstruktor*

**DeList**(DeList *const*& rhs) : counter(0), first(*nullptr*)

{

Iterator it = rhs.begin();

*while* (it != rhs.end())

{

*//Hinzuffügen* *der* *Komponente* *in* *der* *Liste*

*this*->push\_back(\*it);

++it;

}

}

*//Copy* *Zuweisungsoperator*

DeList& *operator*=(DeList *const*& rhs)

{

*//if(this->first* *!=* *rhs.first)*

*if*(*this* != &rhs)

{

*//neustarten*

*this*->clear();

Iterator it = rhs.begin();

*while*(it != rhs.end())

{

*this*->push\_back(\*it);

++it;

}

}

*return* \**this*;

}

void **clear**();

int **size**() *const* {*return* counter;}

Iterator **erase**(Iterator pos);

void **push\_back**(IKomponente\* komp);

Iterator **end**() *const* {*return* Iterator(*nullptr*);}

Iterator **begin**() *const* {*return* Iterator(first);}

*virtual* ~***DeList***() {clear();}

};

***////////////////////////////////////////////////////////////////////////////////***

#include "DeList.h"

#include "IKomponentenElement.h"

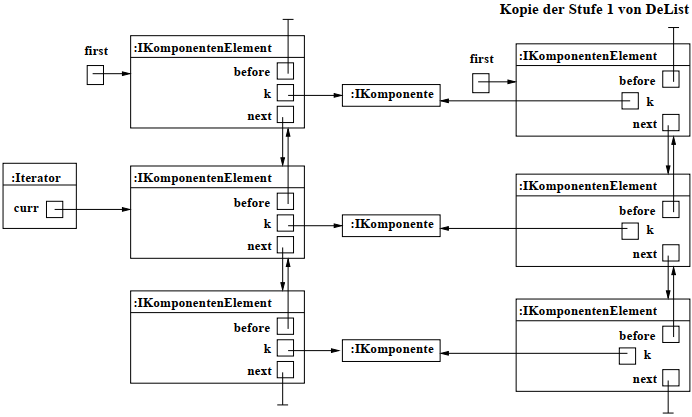
#include "IKomponente.h"

#include "Iterator.h"

***////////////////////////////////////////////////////////////////////////////////***

IKomponentenElement\* DeList::**getElement**(Iterator it) *const*

{

 IKomponentenElement\* tmp = *nullptr*;

*if*(first != *nullptr*)

{

tmp = first;

*while*(tmp != *nullptr*)

{

*if*(\*it == tmp->k)

{

*break*;

}

tmp = tmp->next;

}

}

*return* tmp;

}

***////////////////////////////////////////////////////////////////////////////////***

void DeList::**clear**()

{

Iterator erstes = *this*->begin();

*while* (counter >0)

{

*//Iterator* *erstes* *=* *this->begin();*

*//erase(erstes);*

erstes = erase(erstes);

}

}

***////////////////////////////////////////////////////////////////////////////////***

Iterator DeList::**erase**(Iterator pos)

{

IKomponentenElement\* actual = getElement(pos);

IKomponentenElement\* vorg = *nullptr*;

IKomponentenElement\* nachf = *nullptr*;

Iterator erg;

*if*(actual !=*nullptr*)

{

vorg = actual->before;

nachf = actual->next;

*if* (nachf!=*nullptr*)

{

nachf->before=vorg;

}

*if* (vorg!=*nullptr*)

{

vorg->next=nachf;

}

*else*

{

first=nachf;

}

*delete* actual;

--counter;

erg = Iterator(nachf);

}

*return* erg;

}

***////////////////////////////////////////////////////////////////////////////////***

void DeList::**push\_back**(IKomponente \*komp)

{

IKomponentenElement\* add\_el = *new* IKomponentenElement;

add\_el->k = komp;

add\_el->before = *nullptr*;

add\_el->next = *nullptr*;

*//Wenn* *Liste* *Leer*

*if*(first == *nullptr*)

{

first = add\_el;

}

*else*

{

IKomponentenElement\* last = first;

*while*(last->next != *nullptr*)

{

last = last->next;

}

last->next = add\_el;

add\_el->before = last;

}

++counter;

}

### Werkstueck

#include "IKomponente.h"

#include "Komponente.h"

#include "Iterator.h"

#include "DeList.h"

#include <iostream>

*class* **Werkstueck** : *public* Komponente

{

double height, width;

bool pathIsOptimized;

DeList components;

Iterator **getNearest**(IKomponente *const*\* pk) *const*;

*public*:

**Werkstueck**(double xPos = 0.0, double yPos = 0.0, double h = 1.0, double w = 1.0)

: Komponente(xPos, yPos), height(h), width(w), pathIsOptimized(*false*) {}

double **getHeight**() *const*;

double **getWidth**() *const*;

void **add**(IKomponente\* k);

void **remove**(IKomponente *const*\* k);

IKomponente *const*\* **getChild**(int i) *const*;

double ***calcTotalPath***() *const* *override*;

double **calcPathLength**() *const*;

void **optimizePath**();

void ***output***(std::ostream &os) *const* *override*;

~***Werkstueck***() *override* {}

};

***////////////////////////////////////////////////////////////////////////////////***

#include "Werkstueck.h"

double Werkstueck::**getHeight**() *const*

{

*return* height;

}

***////////////////////////////////////////////////////////////////////////////////***

double Werkstueck::**getWidth**() *const*

{

*return* width;

}

***////////////////////////////////////////////////////////////////////////////////***

Iterator Werkstueck::**getNearest**(*const* IKomponente \*pk) *const*

{

Iterator a = components.begin();

double d = pk->*distance*(\*a);

Iterator erg = a;

++a;

*while* (a != components.end())

{

double tempD = pk->*distance*(\*a);

*if* (tempD < d)

{

d = tempD;

erg = a;

}

++a;

}

*return* erg;

}

***////////////////////////////////////////////////////////////////////////////////***

void Werkstueck::**add**(IKomponente\* k)

{

k->*setParent*(*this*);

components.push\_back(k);

pathIsOptimized = *false*;

}

***////////////////////////////////////////////////////////////////////////////////***

void Werkstueck::**remove**(IKomponente *const*\* k)

{

Iterator it = components.begin();

*while*(it != components.end())

{

*if*(k == \*it)

{

*//* *k->setParent(nullptr);*

(\*it)->*setParent*(*nullptr*);

components.erase(it);

*break*;

}

++it;

}

pathIsOptimized = *false*;

}

***////////////////////////////////////////////////////////////////////////////////***

*const* IKomponente\* Werkstueck::**getChild**(int i) *const*

{

IKomponente\* tmp = *nullptr*;

*if*(i >= 0)

{

Iterator it = components.begin();

*for*(; i > 0 && it != components.end(); --i)

{

++it;

}

tmp = \*it;

}

*return* tmp;

}

***////////////////////////////////////////////////////////////////////////////////***

double Werkstueck::***calcTotalPath***() *const*

{

double erg = 0.0;

Iterator it = components.begin();

*while*(it != components.end())

{

erg += (\*it)->*calcTotalPath*();

++it;

}

erg += *this*->calcPathLength();

*return* erg;

}

***////////////////////////////////////////////////////////////////////////////////***

double Werkstueck::**calcPathLength**() *const*

{

double pathLength = 0.0;

Iterator it = components.begin();

Iterator before = it;

pathLength += *this*->*distance*(\*it);

++it;

*while*(it != components.end())

{

pathLength += (\*before)->*distance*(\*it);

before = it;

++it;

}

pathLength += *this*->*distance*(\*before);

*return* pathLength;

}

***////////////////////////////////////////////////////////////////////////////////***

void Werkstueck::***output***(std::ostream& os) *const*

{

std::string blanks = "";

*const* IKomponente\* tParent = *getParent*();

*while*(tParent != *nullptr*)

{

blanks += " ";

tParent = tParent->*getParent*();

}

os << blanks << "Werkstueck:" << endl;

os << blanks << "Verankerung: (" << *getX*();

os << ", " << *getY*();

os << ")" << endl;

os << blanks << "heigh: " << height;

os << ", width: " << width;

os << endl;

os << blanks << "Komponenten:" << endl;

Iterator it = components.begin();

*while*(it != components.end())

{

*if*(*dynamic\_cast*<Werkstueck\*>(\*it) == *nullptr*)

{

os << blanks;

}

os << \*(\*it);

*if*(*dynamic\_cast*<Werkstueck\*>(\*it) == *nullptr*)

{

os << endl;

}

++it;

}

}

### Main

#include "Bohrung.h"

#include "Fraesung.h"

#include "IKomponente.h"

#include "IKomponentenElement.h"

#include "Komponente.h"

#include "DeList.h"

#include "Werkstueck.h"

#include <iostream>

#include <cmath>

*using* *namespace* std;

int **main**()

{

Bohrung b1(1, 2, 8);

Bohrung b2(2, 6, 4);

Bohrung b3(10, 8, 7);

Bohrung b4(7, 2, 5);

Bohrung b5(1,2,6);

Bohrung b6(5,1,8);

Bohrung b7(5,2.5,8);

Bohrung b8(6,0.5,6);

Fraesung f1(2, 3, 5, sqrt(8), M\_PI\_4);

Fraesung f2(6, 7, 6, 3, 0);

Fraesung f3(17, 6, 4, sqrt(32), M\_PI\_4 + M\_PI);

Fraesung f4(3, 3, 3, sqrt(5), M\_PI - atan((4.00009-3.0)/(3.0-1.00005)));

Fraesung f5(1, 2, 4, sqrt(2), -M\_PI\_4);

Werkstueck w0(0.0, 0.0, 18, 25);

Werkstueck w1(6,10,7,18);

Werkstueck w2(6,3,3,10);

*//W0*

w0.add(&b1);

w0.add(&b2);

w0.add(&b3);

w0.add(&b4);

w0.add(&f1);

w0.add(&f2);

w0.add(&f3);

*//W1*

w1.add(&b5);

w1.add(&b6);

w1.add(&f4);

w0.add(&w1);

*//W2*

w2.add(&b7);

w2.add(&b8);

w2.add(&f5);

w1.add(&w2);

cout << w0 << endl;

cout << "total Path w0: " << w0.*calcTotalPath*() << endl;

cout << "Path length w0: " << w0.calcPathLength() << endl;

cout << "total Path w1: " << w1.*calcTotalPath*() << endl;

cout << "Path length w1: " << w1.calcPathLength() << endl;

cout << "total Path w2: " << w2.*calcTotalPath*() << endl;

cout << "Path length w2: " << w2.calcPathLength() << endl;

w0.optimizePath();

cout << endl << "Optimierung der Pfad-Laenge:" << endl;

cout << w0 << endl;

cout << "total Path w0: " << w0.*calcTotalPath*() << endl;

cout << "Path length w0: " << w0.calcPathLength() << endl;

cout << "total Path w1: " << w1.*calcTotalPath*() << endl;

cout << "Path length w1: " << w1.calcPathLength() << endl;

cout << "total Path w2: " << w2.*calcTotalPath*() << endl;

cout << "Path length w2: " << w2.calcPathLength() << endl;

*return* 0;

}

## P3

### TElement

#include "IKomponente.h"

*template*<*class* T>

*struct* **TElement**

{

T k;

TElement\* before;

TElement\* next;

};

### Iterator

#include "IKomponente.h"

#include "telement.h"

*template* <*typename* T>

*class* **Iterator**

{

TElement<T>\* curr;

*public*:

**Iterator**(TElement<T>\* currPos = *nullptr*)

:curr(currPos) {}

Iterator& *operator*++()

{

*if*(curr != *nullptr*)

{

curr = curr->next;

}

*return* \**this*;

}

bool *operator*!=(Iterator *const*& it) *const*

{

*return* *this*->curr != it.curr;

}

bool *operator*==(Iterator *const*& it) *const*

{

*return* *this*->curr == it.curr;

}

T& *operator*\*() *const*;

*virtual* ~***Iterator***() {}

};

*template*<*class* T>

T& Iterator<T>::*operator*\*() *const*{

*return* curr->k;

}

### DeList

#include "Komponente.h"

#include "telement.h"

#include "IKomponente.h"

#include "Iterator.h"

*template*<*class* T>

*class* **DeList**

{

int counter;

TElement<T>\* first;

TElement<T>\* **getElement**(Iterator<T> it) *const*;

*public*:

**DeList**() : counter(0), first(*nullptr*) {}

*//copy* *Konstruktor*

**DeList**(DeList *const*& rhs) : counter(0), first(*nullptr*)

{

Iterator<T> it = rhs.begin();

*while* (it != rhs.end())

{

*this*->push\_back(\*it);

++it;

}

}

*//Copy* *Zuweisungsoperator*

DeList& *operator*=(DeList *const*& rhs)

{

*if*(*this* != &rhs)

{

*//neustarten*

*this*->clear();

Iterator<T> it = rhs.begin();

*while*(it != rhs.end())

{

*this*->push\_back(\*it);

++it;

}

}

*return* \**this*;

}

void **clear**();

int **size**() *const* {*return* counter;}

Iterator<T> **erase**(Iterator<T> pos);

void **push\_back**(T komp);

Iterator<T> **end**() *const* {*return* Iterator<T>(*nullptr*);}

Iterator<T> **begin**() *const* {*return* Iterator<T>(first);}

*virtual* ~***DeList***() {clear();}

};

***////////////////////////////////////////////////////////////////////////////////***

*template*<*class* T>

TElement<T>\* DeList<T>::**getElement**(Iterator<T> it) *const*

{

TElement<T>\* tmp = *nullptr*;

*if*(first != *nullptr* && it != end())

{

tmp = first;

*while*(tmp != *nullptr*)

{

*if*(\*it == tmp->k)

{

*break*;

}

tmp = tmp->next;

}

}

*return* tmp;

}

***////////////////////////////////////////////////////////////////////////////////***

*template* <*class* T>

void DeList<T>::**clear**()

{

Iterator<T> erstes = *this*->begin();

*while* (counter >0)

{

erstes = erase(erstes);

}

}

***////////////////////////////////////////////////////////////////////////////////***

*template* <*class* T>

Iterator<T> DeList<T>::**erase**(Iterator<T> pos)

{

TElement<T>\* actual = getElement(pos);

TElement<T>\* vorg = *nullptr*;

TElement<T>\* nachf = *nullptr*;

Iterator<T> erg;

*if*(actual !=*nullptr*)

{

vorg = actual->before;

nachf = actual->next;

*if* (nachf!=*nullptr*)

{

nachf->before=vorg;

}

*if* (vorg!=*nullptr*)

{

vorg->next=nachf;

}

*else*

{

first=nachf;

}

*delete* actual;

--counter;

erg = Iterator<T>(nachf);

}

*return* erg;

}

***////////////////////////////////////////////////////////////////////////////////***

*template*<*class* T>

void DeList<T>::**push\_back**(T komp)

{

TElement<T>\* add\_el = *new* TElement<T>;

add\_el->k = komp;

add\_el->before = *nullptr*;

add\_el->next = *nullptr*;

*//Wenn* *Liste* *Leer*

*if*(first == *nullptr*)

{

first = add\_el;

}

*else*

{

TElement<T>\* last = first;

*while*(last->next != *nullptr*)

{

last = last->next;

}

last->next = add\_el;

add\_el->before = last;

}

++counter;

}

### Werkstueck

#include "IKomponente.h"

#include "Komponente.h"

#include "Iterator.h"

#include "DeList.h"

#include <QJsonDocument>

#include <QJsonObject>

#include <QJsonArray>

#include <QFile>

#include <iostream>

*class* **Werkstueck** : *public* Komponente

{

double height, width;

bool pathIsOptimized;

DeList<IKomponente\*> components;

Iterator<IKomponente\*> **getNearest**(IKomponente *const*\* pk) *const*;

*public*:

**Werkstueck**(double xPos = 0.0, double yPos = 0.0, double h = 1.0, double w = 1.0)

: Komponente(xPos, yPos), height(h), width(w), pathIsOptimized(*false*) {}

double **getHeight**() *const*;

double **getWidth**() *const*;

void **add**(IKomponente\* k);

void **remove**(IKomponente *const*\* k);

IKomponente *const*\* **getChild**(int i) *const*;

double ***calcTotalPath***() *const* *override*;

double **calcPathLength**() *const*;

void **optimizePath**();

QJsonObject ***toJson***() *const* *override*;

void ***output***(std::ostream &os) *const* *override*;

~***Werkstueck***() *override* {}

};

***////////////////////////////////////////////////////////////////////////////////***

#include "Werkstueck.h"

double Werkstueck::**getHeight**() *const*{*return* height;}

***////////////////////////////////////////////////////////////////////////////////***

double Werkstueck::**getWidth**() *const*{*return* width;}

***////////////////////////////////////////////////////////////////////////////////***

Iterator<IKomponente\*> Werkstueck::**getNearest**(*const* IKomponente \*pk) *const*

{

Iterator<IKomponente\*> a = components.begin();

double d = pk->*distance*(\*a);

Iterator<IKomponente\*> erg = a;

++a;

*while* (a != components.end())

{

double tempD = pk->*distance*(\*a);

*if* (tempD < d)

{

d = tempD;

erg = a;

}

++a;

}

*return* erg;

}

***////////////////////////////////////////////////////////////////////////////////***

void Werkstueck::**add**(IKomponente\* k)

{

k->*setParent*(*this*);

components.push\_back(k);

pathIsOptimized = *false*;

}

***////////////////////////////////////////////////////////////////////////////////***

void Werkstueck::**remove**(IKomponente *const*\* k)

{

Iterator<IKomponente\*> it = components.begin();

*while*(it != components.end())

{

*if*(k == \*it)

{

(\*it)->*setParent*(*nullptr*);

components.erase(it);

*break*;

}

++it;

}

pathIsOptimized = *false*;

}

***////////////////////////////////////////////////////////////////////////////////***

*const* IKomponente\* Werkstueck::**getChild**(int i) *const*

{

IKomponente\* tmp = *nullptr*;

*if*(i >= 0 && i < components.size())

{

Iterator<IKomponente\*> it = components.begin();

*for*(; i > 0 && it != components.end(); --i)

{

++it;

}

tmp = \*it;

}

*return* tmp;

}

***////////////////////////////////////////////////////////////////////////////////***

double Werkstueck::***calcTotalPath***() *const*

{

double erg = 0.0;

Iterator<IKomponente\*> it = components.begin();

*while*(it != components.end())

{

erg += (\*it)->*calcTotalPath*();

++it;

}

erg += *this*->calcPathLength();

*return* erg;

}

***////////////////////////////////////////////////////////////////////////////////***

double Werkstueck::**calcPathLength**() *const*

{

double pathLength = 0.0;

Iterator<IKomponente\*> it = components.begin();

Iterator<IKomponente\*> before = it;

*if*(it != components.end())

{

pathLength += *this*->*distance*(\*it);

++it;

}

*while*(it != components.end())

{

pathLength += (\*before)->*distance*(\*it);

before = it;

++it;

}

*if*(before != components.end())

{pathLength += *this*->*distance*(\*before);}

*return* pathLength;

}

***////////////////////////////////////////////////////////////////////////////////***

void Werkstueck::***output***(std::ostream& os) *const*

{

std::string blanks = "";

*const* IKomponente\* tParent = *getParent*();

*while*(tParent != *nullptr*)

{

blanks += " ";

tParent = tParent->*getParent*();

}

os << blanks << "Werkstueck:" << endl;

os << blanks << "Verankerung: (" << *getX*();

os << ", " << *getY*();

os << ")" << endl;

os << blanks << "heigh: " << height;

os << ", width: " << width;

os << endl;

os << blanks << "Komponenten:" << endl;

Iterator<IKomponente\*> it = components.begin();

*while*(it != components.end())

{

*if*(*dynamic\_cast*<Werkstueck\*>(\*it) == *nullptr*)

{

os << blanks;

}

os << \*(\*it);

*if*(*dynamic\_cast*<Werkstueck\*>(\*it) == *nullptr*)

{

os << endl;

}

++it;

}

}

***////////////////////////////////////////////////////////////////////////////////***

QJsonObject Werkstueck::***toJson***() *const*

{

QJsonArray temp;

Iterator<IKomponente\*> it;

*for*(it = components.begin(); it!=components.end(); ++it)

{

temp.push\_back((\*it)->*toJson*());

}

QJsonObject result = Komponente::toJson();

result["components"] = temp;

result["height"] = height;

result["width"] = width;

result["pathIsOptimized"] = pathIsOptimized;

result["type"] = "Werkstueck";

*return* result;

}

### Serializer

#include "IKomponente.h"

*class* **Serializer**

{

IKomponente *const*& ik;

*public*:

**Serializer**(IKomponente *const*& root)

: ik(root) {}

void **writeToJson**(*const* char\* fname) *const*;

*virtual* ~***Serializer***() {}

};

#include "Serializer.h"

#include <QJsonDocument>

#include <QJsonObject>

#include <QFile>

#include <fstream>

#include <stdexcept>

void Serializer::**writeToJson**(*const* char \*fname) *const*

{

QJsonDocument doc(ik.*toJson*());

QFile out\_file(fname);

*if*(!out\_file.*open*(QFile::*WriteOnly*))

{

*throw* std::fstream::failure("Datei konnte nicht geoffnet werden!");

}

*if*(out\_file.write(doc.toJson()) == -1)

{

*throw* std::fstream::failure("Datei konnte nicht beschrieben werden!");

}

out\_file.*close*();

}

### Komponenten Änderungen

#include <QJsonDocument>

*class* **Komponente** : *public* IKomponente

{

*/\*.................\** */*

QJsonObject ***toJson***() *const* *override*;

};

QJsonObject Komponente::***toJson***() *const*

{

QJsonObject result;

result["type"] = "Komponente";

result["hasParent"] = (parent != *nullptr*);

result["x"] = x;

result["y"] = y;

*return* result;

}

### Bohrung Änderungen

#include <QJsonDocument>

#include <QJsonObject>

#include <QFile>

*class* **Bohrung** : *public* Komponente

{

*/\*.................\** */*

QJsonObject ***toJson***() *const* *override*;

};

QJsonObject Bohrung::***toJson***() *const*

{

QJsonObject result = Komponente::toJson();

result["diameter"] = diameter;

result["type"] = "Bohrung";

*return* result;

}

### Fraesung Änderungen

#include <QJsonDocument>

#include <QJsonObject>

#include <QFile>

*class* **Fraesung** : *public* Bohrung

{

*/\*.................\** */*

QJsonObject ***toJson***() *const* *override*;

};

QJsonObject Fraesung::***toJson***() *const*

{

QJsonObject result = Bohrung::toJson();

result["angle"] = getAngle();

result["length"] = getLength();

result["type"] = "Fraesung";

*return* result;

}

### Main Änderungen

int **main**()

{

Serializer jsonfile = Serializer(w0);

*try* {

jsonfile.writeToJson("w0.json");

} *catch* (*const* std::fstream::failure& e) {

std::cerr<< e.*what*() << endl;

} *catch* (...){

cout<< "Eine Exception ausgeloest"<<endl;

}

*return* 0;

}