

```

1: #pragma once
2: #include <cassert> // static_assert
3: #include <cmath> // sqrt()
4: #include <iostream>
5: #include <type_traits> // std::is_floating_point<T>()
6:
7: //using namespace std; // don't use in header
8:
9: /** Template Klasse fuer komplexe Zahlen zur Demonstration.
10:  * @warning Bei Templateklassen (und -funktionen) muss der Implementierungsteil,
11:  * hier im File komplex.cpp, im Headerfile inkludiert werden.
12:  * Das Implementierungsfile darf nicht noch einmal ueberstzt werden!
13:  */
14: template <class T>
15: class Komplex
16: {
17:     // check type of template at compile time
18:     static_assert(std::is_floating_point<T>(), "Vector elements have to be floating point numbers."
); // C++11
19: public:
20:     /** Default constructor initializes both components by zero. */
21:     Komplex() : Komplex(T(0), T(0)) // constructor forwarding
22:     { }
23:
24:     /** Parameter constructor
25:      * \param[in] re Realteil
26:      * \param[in] im Imaginaerteil (default value: 0.0)
27:      */ // -- Standardwert
28:     Komplex(T re, T im=0.0) // Parameterkonstruktor mit ein oder zwei Argumenten
29:     : _re(re), _im(im)
30:     { }
31:     // See rule of five: https://en.cppreference.com/w/cpp/language/rule_of_three
32:     //
33:     Komplex(const Komplex<T>& org) = default; // Copykonstruktor
34:     Komplex(Komplex<T>&& org) = default; // Movekonstruktor
35:     Komplex<T>& operator=(const Komplex<T>& rhs) = default; // Copy-Zuweisungsoperator
36:     Komplex<T>& operator=(Komplex<T>&& rhs) = default; // Move-Zuweisungsoperator
37:     ~Komplex() = default; // Destruktor
38:
39:     /** Abfrage des Realteils
40:      * \return Realteil

```

Komplex <string> a(...)

abgeleitet von Compilerzeit

zur Compilezeit

$T = \text{float, double, long double}$

```
41:      */
42:      //      |-- Member der Instanz werden durch die Methode nicht veraendert!
43:      T Get_re() const
44:      {
45:          return _re;
46:      }
47:      /** Abfrage des Realteils
48:       * \param[in] val New value to set
49:       */
50:      void Set_re(T val)
51:      {
52:          _re = val;
53:      }
54:      /** Abfrage des Imaginaerteils
55:       * \return Imaginaerteil
56:       */
57:      //      |-- Member der Instanz werden durch die Methode nicht veraendert!
58:      T Get_im() const
59:      {
60:          return _im;
61:      }
62:      /** Setzen des Realteils
63:       * \param[in] val New value to set
64:       */
65:      void Set_im(T val)
66:      {
67:          _im = val;
68:      }
69:
70:      /** Addiert zur aktuellen Instanz eine zweite komplexe Zahl
71:       * \param[in] rhs zweite komplexe Zahl
72:       * \return \p *this += \p rhs
73:       */
74:      Komplex<T>& operator+=(const Komplex<T>& rhs);
75:
76:      /** Addiert die aktuelle Instanz mit einer zweiten komplexen Zahl
77:       * \param[in] rhs zweite komplexe Zahl
78:       * \return \p *this + \p rhs
79:       */
80:      //      |-- Member der Instanz werden durch die Method
e nicht veraendert!
```

```
81:     Komplex<T> operator+(const Komplex<T>& rhs) const;
82:
83:     bool operator<(const Komplex<T>& rhs) const
84:     {
85:         return _re < rhs._re || ( _re == rhs._re && _im < rhs._im );
86:     }
87:
88:     bool operator==(const Komplex<T>& rhs) const
89:     {
90:         return _re == rhs._re && _im == rhs._im ;
91:     }
92:
93:     bool operator>(const Komplex<T>& rhs) const
94:     {
95:         return !( *this < rhs || *this== rhs ) ;
96:     }
97:
98:
99: protected:
100: private:
101:     T _re; //!< Realteil
102:     T _im; //!< Imaginaerteil
103: };
104:
105: /** Ausgabeoperator fuer die Klasse.
106:  * \param[in] s ein beliebiger Ausgabestrom
107:  * \param[in] rhs die auszugebende Instanz
108:  */
109: template <class T>
110: std::ostream& operator<<(std::ostream& s, const Komplex<T>& rhs);
111:
112:
113:     /** Addiert zu einer reellen Zahl eine komplexe Zahl
114:     * \param[in] lhs reelle Zahl
115:     * \param[in] rhs komplexe Zahl
116:     * \return \p lhs + \p rhs
117:     */
118: template <class T>
119: Komplex<T> operator+(T lhs, const Komplex<T>& rhs);
120:
121:
```

```
122: template <class T>
123: T abs(const Komplex<T>& rhs)
124: {
125:     return std::sqrt(rhs.Get_re()*rhs.Get_re()+rhs.Get_im()*rhs.Get_im());
126: }
127:
128:
129: #include "komplex.tcc"
130:
```

```
1:  ///#include "komplex.h"
2:
3:  template <class T>
4:  Komplex<T>& Komplex<T>::operator+=(const Komplex<T>& rhs)
5:  {
6:      _re += rhs._re;
7:      _im += rhs._im;
8:      return *this; // this ist ein Pointer auf die aktuelle Instanz
9:  }
10:
11: template <class T>
12: Komplex<T> Komplex<T>::operator+(const Komplex<T>& rhs) const
13: {
14:     Komplex<T> tmp(*this);
15:     return tmp+=rhs;
16: }
17:
18: template <class T>
19: std::ostream& operator<<(std::ostream& s, const Komplex<T>& rhs)
20: {
21:     s << "(" << rhs.Get_re() << ", " << rhs.Get_im() << ")";
22:     return s;
23: }
24:
25: template <class T>
26: Komplex<T> operator+(T lhs, const Komplex<T>& rhs)
27: {
28:     return rhs+lhs; // Ruft Methode operator+ der Klasse Komplex
29:
30: }
```

```
1: // Klasse Komplex wird erweitert; Templates
2: // operator+ wird aus operator += abgeleitet
3: // Vergleichsoperatoren: <, == und daraus abgeleitet >
4: // ( nur zur Demo: Vergleichsoperatoren fuer komplexe Zahlen sind nicht transitiv !!)
5: #include "komplex.h"
6: #include <algorithm>           // copy, sort
7: #include <iostream>
8: #include <iterator>           // ostream_iterator
9: #include <vector>
10: using namespace std;
11:
12: template <class T>
13: ostream &operator<<(ostream &s, const vector<T> &v)
14: {
15:     // for (auto it=v.begin(); it!=v.end; ++it) cout << *it << " ";
16:     copy(v.begin(), v.end(), ostream_iterator<T, char>(s, " "));
17:     return s;
18: }
19:
20: template <class T>
21: bool islargerequal(T a, T b)
22: {
23:     return !(a < b);
24: }
25:
26: int main()
27: {
28:     const Komplex<double> a(3.2, -1.1); // Konstruktor Komplex(double,double)
29:     const Komplex<double> b(4, -1);     // Konstruktor Komplex(double,double)
30:     Komplex<double> c;                  // Konstruktor Komplex() wird benoetigt
31:
32:     c = a + b;                          // OK: a.operator+(const Komplex&)
33:
34:     cout << a << endl;                  // Ausgabeoperator
35:     cout << c << endl;
36:
37:     Komplex<double> dd(-3.2);
38:     dd += a;                             // OK: a.operator+(const Komplex&)
39:     cout << dd << endl;
40:
41:     cout << (dd < a) << endl;
```

```
42:     cout << (dd == a) << endl;
43:     cout << (dd > a) << endl;
44:
45:     vector<Komplex<float>> vv = { {3.0F, -1.0F}, {3.0F, -3.0F}, {1.2F, -4.0F}, {4.3F, -1.0F} };
46:     cout << "vv : " << vv << endl;
47:     sort(vv.begin(), vv.end()); // requires operator<, ans operator= (for vector-container)
48:     cout << "vv : " << vv << endl;
49:
50:     sort(vv.begin(), vv.end(), islargerequal<Komplex<float>>());
51:     cout << "vv : " << vv << endl;
52:
53:     // order wrt. abs(), with lambda function
54:     sort(vv.begin(), vv.end(), //islargerequal<Komplex<float>>
55:         [] (auto const &aa, auto const &bb) -> bool
56:         { return abs(aa) < abs(bb); }
57:     );
58:     cout << "vv: abs : " << vv << endl;
59:
60:
61:     auto it = find(vv.begin(), vv.end(), Komplex<float>(1.22F, -4.0F) );
62:     if (it != vv.end()) {cout << " found " << *it << endl;}
63:
64:     Komplex<long double> lda(1.22L, -4.0L); cout << lda << endl;
65:     //Komplex<int> ia(-1,2); cout << ia << endl;
66:
67:     // https://stackoverflow.com/questions/53557649/how-do-i-check-for-c20-support-what-is-the-value-of-cplusplus-for-c20
68:     #if __cplusplus >= 202002L
69:         cout << " C++20 support" << endl;
70:     #else
71:         cout << " C++ version: " << __cplusplus << endl;
72:     #endif
73:
74:
75:     return 0;
76: }
```