Datentypen für komplexe Zahlen in der Standardbibliothek (`<complex>`)

Vordefiniert sind die Datentypen `complex<float>`, `complex<double>`, `complex<long double>`.

Im folgenden stehen x, y für relle Zahlen vom Typ T und w, z für komplexwertige Zahlen vom Typ `complex<`T`>`, worin T `float`, `double` oder `long double` bezeichnet.

Funktion	*Wirkung*
`complex<T> z`	vereinbart z mit Wert 0
`complex<T> z(x)`	vereinbart z mit Wert x
`complex<T> z(x,y)`	vereinbart z mit Wert $x+iy$
`+ - * /`	arith. Grundoperationen (unär,binär)
`= += -= *= /=`	Zuweisungen
`== !=`	Vergleiche
`cin >> z`	Eingabe im Format x, (x), (x,y)
`cout << z`	Ausgabe im Format (x,y)
`z.real() z.imag()`	Re z, Im z
`real(z) imag(z) conj(z)`	Re z, Im z, $\bar{z}$
`abs(z) norm(z)`	$\lvert z\rvert$, $\lvert z\rvert^2$
`arg(z)`	`atan2(imag(z),real(z))`
`polar(r,phi)`	$re^{i\varphi}$
`sin(z) cos(z) tan(z)`	trig. Funktionen
`exp(z) sinh(z) cosh(z) tanh(z)`	e^z, hyberbol. Funkt.
`log(z)`	$\ln z$, Verzweig. in $(-\infty,0)$, $\ln(x\pm 0\cdot i)=\ln\lvert x\rvert \pm \pi\cdot i$ $(x<0)$
`log10(z)`	$\ln z/\ln 10$, ln wie vorige Zeile
`pow(z,w)`	$\exp(w\ln z)$, ln wie oben
`sqrt(z)`	$\sqrt{z}$, Verzweig. in $(-\infty,0)$, $\sqrt{x\pm 0\cdot i}=\pm\sqrt{\lvert x\rvert}\cdot i$ $(x<0)$

Beispiel:

```
#include <iostream>
#include <complex>
#include <cmath>
#include <limits>

using namespace std;

int main()
{
  complex<double> z, i(0.0,1.0);
  cout.precision(numeric_limits<double>::digits10);

  cout << "(Re,Im): ";
  cin >> z;
  cout << "sqrt(z)     = " << sqrt(z)        << endl
       << "sqrt(1.0+i) = " << sqrt(1.0+i)    << endl;
  return 0;
}
```

Bildschirmausgabe:

```
(Re,Im): (2,3)
sqrt(z)     = (1.67414922803554,0.895977476129838)
sqrt(1.0+i) = (1.09868411346781,0.455089860562227)
```

Voreinstellungen für Initialisierung und Zuweisung

Voreingestellt ist folgendes Verhalten für Klassen: Initialisierungen durch ein anderes Klasssenobjekt - dazu gehören auch Parameterübergaben und die Rückgabe von Funktionswerten - und Zuweisungen erfolgen jeweils komponentenweise.

Bsp.: Polynomklasse (rudimentär)

```
                :
class Polynom {
  private:
    vector<double> a;

  public:
    Polynom() { }                                  // Nullpolynom (Laenge 0, Grad:=-1)
    Polynom(int n): a(n+1) { a[n]=1; }             // Monom x^n
    Polynom(const vector<double>& v): a(v) { }  // Polynom initial. entspr. Vektor v

    int grad() { return a.size()-1; }

    double value(double x)
    {
      double s=0, xpot=1;
      for (int i=0; i<a.size(); ++i) {
        s += a[i]*xpot;
        xpot *= x;
      }
      return s;
    }
};


int main()
{
  vector<double> v;
  double x;

  // Einlesen des Koeffizientenvektors und von x
  //            :

  Polynom r(v),p;
  p = r;
  cout << "p(x) = " << p.value(x) << endl;

  return 0;
}
```

Ressourcenallokation und -deallokation

Benötigt eine Klasse zusätzliche Ressourcen (dynamischer Speicher, Dateizugriff), so wird die Ressourcenallokation oft auch in einem oder mehreren Konstruktoren vorgenommen. Zur Vermeidung unnötig belegter Ressourcen ist in der Regel ein Destruktor erforderlich, der die Deallokation ausführt.

Imperative Programmierung in C, C++ und Java (Auszug)

C++ und Java wurden mit dem Ziel entwickelt, objektorientierte Programmierung zu unterstützen. Imperative Programmierung ist in C++ einfach möglich, weil C eine imperative Sprache ist und C++ weitgehend abwärtskompatibel zu C ist. In C++ existieren neben den aus C übernommenen Sprachelementen teilweise weitere – besser an die objektorientierte Programmierung angepasste – mit ähnlicher Funktionalität. (Vektoren, Strings, Ein/Ausgabe, dynamische Speicherverwaltung). In Java ist imperative Programmierung nur eingeschränkt möglich.

Headerdateien, Programmaufbau

C: Die Standardheaderdateien haben den Namen: header.h. Das gilt ebenso für C++92. Das Hauptprogramm wird von der Funktion `main` bereitgestellt.

C++: C-Headerdateien sind ab C++98 umbenannt: header.h→cheader
Es gibt weitere Headerdateien zur Benutzung der Standard Template Library (STL)
Geringerer Schreibaufwand durch die Namespace-Direktive: `using namespace std;`
Das Hauptprogramm wird von der Funktion `main` bereitgestellt.

Java: Importdeklarationen, z.B. `import java.util.*`, treten an die Stelle von `#include`-Anweisungen. Mindestens eine Klasse muss eine statische Methode mit Namen `main` enthalten.

Übersetzen, Starten (Ubuntu Linux 20.04)

C: Übersetzen und Starten (GNU-Compiler, Linken der Math.bibl. `libm.so` erforderlich):
```
cc -Wall prog.c -lm
./a.out
```

C++: Übersetzen und Starten (GNU-Compiler, `-lm` nicht erforderlich):
```
c++ -Wall prog.cpp
./a.out
```

Java: Ein Javaprogramm wird zuerst in Java-Bytecode übersetzt und dann mit einem javaspezifischen Startprogramm aufgerufen (Oracle-Java):
```
javac -Xlint Klasse.java
java Klasse
```
Der Dateiname *ohne* die Endung `.java` muss mit dem Klassennamen übereinstimmen!

Variablen- und Funktionsdefinitionen

C: In ANSI-C sind Variablendefinitionen nur am Blockanfang und außerhalb von Funktionen möglich, in C99 überall. Funktionsdefinitionen sind nur auf oberster Ebene zulässig.

C++: Variablenvereinbarungen sind überall möglich, Funktionsdefinitionen auf oberster Ebene und in Klassen.

Java: Variablen und Funktionsdefinitionen sind nur in Klassen möglich. Für die imperative Programmierung müssen daher statische Funktionen benutzt werden. Java verlangt und überprüft die Initialisierung aller Variablen.

Klassen

Klassen können als Verallgemeinerung von Records angesehen werden. Es handelt sich dabei um Datentypen, in denen neben Datenkomponenten zusätzlich Funktionen vereinbart und Zugriffsattribute angegeben werden können.

C: In C gibt es nur Records (`struct`). Dem Recordnamen ist immer `struct` voranzustellen.

C++: Klassen können sowohl als `class` oder `struct` vereinbart werden, die Voreinstellung für die Zugriffsrechte unterscheidet sich aber. Die Klassennamen werden direkt als Typnamen verwendet. In Klassen können Komponentenfunktionen (Konstruktoren, Destruktoren, statische und nichtstatische Komponentenfunktionen) und befreundete Funktionen vorkommen. Komponentenfunktionen können virtuell sein (Voreinstellung: nicht virtuell). Einfach- und Mehrfachvererbung sind möglich, ebenso Überladen von Operatoren und Templates.

Java: Klassen werden mit `class` vereinbart und ihr Name direkt als Typname benutzt. In Java-Klassen gibt es weder befreundete Funktionen noch Destruktoren, letztere Aufgabe übernimmt der Garbage-Collector. Die Komponentenfunktionen (Methoden) sind virtuell, sofern nicht anders vereinbart. Nur Einfachvererbung ist möglich.

Zeiger und Referenzen

C:
```
C c,d; c=d;         // c,d verschiedene Speicherplaetze, gleicher Wert
C *cp,*dp; cp=dp;   // *cp,*dp identische Speicherplaetze
C *cp = malloc(sizeof(C)) // dynamisch allozierter Speicherplatz
```
Es gibt keine Referenzen, im Bedarfsfall muss auf Zeiger zurückgegriffen werden.

C++:
```
C c,d; c=d;         // c,d verschiedene Speicherplaetze, gleicher Wert
C *cp,*dp; cp=dp;   // *cp,*dp identische Speicherplaetze
C *cp = new C;      // *cp dynamisch allozierter Speicherplatz
C& e=c;             // e Referenz auf c: identische Speicherplaetze
```
Realisierung von Referenzen durch Zeiger möglich: Compiler ersetzt die letzte Anweisung durch `C *ep=c` und überall sonst `e` durch `*ep`.

Zeiger und Referenzen sind in C++ sowohl für einfache Datentypen als auch für Klassen möglich. Für die Zuweisung und die Nichtreferenzparameterübergabe ist bei den Standardklassen Wertsemantik üblich.

Java: Es gibt keine Zeiger. Für Klassen und Vektoren werden immer Referenzen eingesetzt. Die Syntax unterscheidet sich von C++:
```
C c = new C, d=c;  // c,d identische Speicherplaetze!
```
Für Kopien muss in der Klasse eine Methode `clone` definiert sein:
```
C e = c.clone();   // c,e verschiedene Speicherplaetze, gleicher Wert
```
Bei der Zuweisung einfacher Datentypen erfolgt wie in C und C++ eine Kopie des Werts.

Parameterübergabe, Referenzparameter

C: Nur Wertparameterübergabe, außer für C-Vektoren (dort: Übergabe der Adresse der Komponente 0). Referenzparameterübergabe muss mittels Zeigern realisiert werden.

C++: Wertparameterübergabe und Referenzparameterübergabe sowohl für einfache Datentypen als auch für Klassen möglich. C-Vektoren werden wie in C übergeben.

Java: Nur Wertparameterübergabe für einfache Datentypen und nur Referenzparameterübergabe für Klassen und Vektoren möglich.