Ludwig-Maximilians-Universität München
Mathematisches Institut
Gregor Kleen

Programmieren II für Studierende der Mathematik

Blatt 1 – Lösungsvorschlag

Aufgabe 1 Erstellen Sie eine Klasse `Complex`, die komplexe Zahlen $z = r \cdot e^{i\varphi}$, $r \geq 0$, $\varphi \in [-\pi, \pi]$, intern mithilfe der Datenkomponenten `r` und `phi` speichert. Auch für `r` und `phi` soll sichergestellt werden, dass die obigen Invarianten stets gelten.

Um mit sinnvoller Kapselung sicherzustellen, dass die Invarianten erfüllt sind sollten `r` und `phi` auf jeden Fall `private` sein. Dies verhindert, dass mit der Klasse `Complex` nicht assoziierter Code durch direkten Zugriff auf die Attribute die Invariante beliebig verletzen kann.

Klasse

```
class Complex {
  private:
    double r, phi;

  public:
    Konstruktor

    Arithmetik

    Funktionen

    Ausgabe
};
```

Der Konstruktor soll Argumente `x` und `y` akzeptieren und die komplexe Zahl $z = x + iy$ erzeugen (d.h. $r = \sqrt{x^2 + y^2}$ und $\varphi = \operatorname{atan2}(y, x)$). Sind `x`, `y` oder beide nicht angegeben, so soll stattdessen jeweils `0` verwendet werden. D.h. den Konstruktor ohne Argumente aufzurufen soll die komplexe Zahl 0 anlegen.

Konstruktor

```
Complex(double Re=0, double Im=0):
  r(sqrt(Re*Re + Im*Im)), phi(atan2(Im, Re)) {}
```

Überladen Sie die arithmetischen Operatoren `+`, `-`, `*` und `/` in mathematisch sinnvoller Weise.

Hinweis. Für $z_1 = r_1 \cdot e^{i\varphi_1}$ und $z_2 = r_2 \cdot e^{i\varphi_2}$ gilt mit $k \in \mathbb{Z}$ jeweils geeignet:

$$z_1 \pm z_2 = r \cdot e^{i\varphi} \quad \text{mit} \quad r = \sqrt{r_1^2 + r_2^2 \pm 2 \cdot r_1 r_2 \cos(\varphi_1 - \varphi_2)}$$
$$\varphi = \operatorname{atan2}(r_1 \sin(\varphi_1) \pm r_2 \sin(\varphi_2), r_1 \cos(\varphi_1) \pm r_2 \cos(\varphi_2))$$

$$z_1 \cdot z_2 = r \cdot e^{i\varphi} \quad \text{mit} \quad r = r_1 \cdot r_2$$
$$\varphi = \varphi_1 + \varphi_2 + k \cdot 2\pi \in [-\pi, \pi]$$

$$\frac{z_1}{z_2} = r \cdot e^{i\varphi} \quad \text{mit} \quad r = \frac{r_1}{r_2}$$
$$\varphi = \varphi_1 - \varphi_2 + k \cdot 2\pi \in [-\pi, \pi]$$

In den Fällen `*` und `/` gilt, wegen der Invariante für `phi`, jeweils $|\varphi_1 \pm \varphi_2| \leq 2\pi$. Es reicht daher aus geeignet $\pm 2\pi$ zu addieren um wieder in den gewünschten Bereich zu kommen.

Arithmetik

```
friend Complex operator+(Complex z1, Complex z2) {
  Complex z;
  z.r = sqrt(z1.r*z1.r + z2.r*z2.r + 2*z1.r*z2.r*cos(z1.phi - z2.phi));
  z.phi = atan2(z1.r*sin(z1.phi) + z2.r*sin(z2.phi),
                z1.r*cos(z1.phi) + z2.r*cos(z2.phi));
  return z;
}

friend Complex operator-(Complex z1, Complex z2) {
  Complex z;
  z.r = sqrt(z1.r*z1.r + z2.r*z2.r - 2*z1.r*z2.r*cos(z1.phi - z2.phi));
  z.phi = atan2(z1.r*sin(z1.phi) - z2.r*sin(z2.phi),
                z1.r*cos(z1.phi) - z2.r*cos(z2.phi));
  return z;
}

friend Complex operator*(Complex z1, Complex z2) {
  Complex z;
  z.r = z1.r * z2.r;
  z.phi = z1.phi + z2.phi;

  if (z.phi >  M_PI) z.phi -= 2*M_PI;
  if (z.phi < -M_PI) z.phi += 2*M_PI;

  return z;
}

friend Complex operator/(Complex z1, Complex z2) {
  Complex z;
  z.r = z1.r / z2.r;
  z.phi = z1.phi - z2.phi;

  if (z.phi >  M_PI) z.phi -= 2*M_PI;
  if (z.phi < -M_PI) z.phi += 2*M_PI;

  return z;
```

```
}
```

Erstellen Sie zudem für die Exponentialfunktion (`exp`), den Hauptwert der Quadratwurzel (`sqrt`), den Logarithmus (`log`) und die allgemeine Potenzfunktion (`pow`) jeweils entsprechende Funktionen.

Hinweis. Für $z = r \cdot e^{i\varphi}$ gilt mit $k \in \mathbb{Z}$ jeweils geeignet:

$$\exp(z) = r' \cdot e^{i\varphi'} \qquad \text{mit} \quad r' = e^{r\cos(\varphi)}, \quad \varphi' = r\sin(\varphi) + k \cdot 2\pi \in [-\pi, \pi]$$

$$\sqrt{z} = r' \cdot e^{i\varphi'} \qquad \text{mit} \quad r' = \sqrt{r}, \quad \varphi' = \frac{\varphi}{2}$$

$$\log(z) = \ln(r) + i\varphi$$

Im Fall `exp` muss φ in den gewünschten Bereich gebracht werden. Hierfür verwenden wir zunächst die Funktion `fmod`; diese berechnet den Rest der Division zweier Gleitpunktzahlen wiederum als Gleitpunktzahl. Um $\varphi \in [-2\pi, 2\pi]$ wieder in den gewünschten Bereich zu bringen, reicht es dann wieder $\pm 2\pi$ zu addieren. Alternativ, aber weniger performant, könnte auch eine Schleife verwendet werden.

Funktionen

```
friend Complex sqrt(Complex z1) {
  Complex z;
  z.r = sqrt(z1.r);
  z.phi = z1.phi / 2;
  return z;
}

friend Complex exp(Complex z1) {
  Complex z;
  z.r = exp(z1.r * cos(z1.phi));
  z.phi = z1.r * sin(z1.phi);

  z.phi = fmod(z.phi, 2*M_PI);
  if (z.phi >  M_PI) z.phi -= 2*M_PI;
  if (z.phi < -M_PI) z.phi += 2*M_PI;

  return z;
}

friend Complex log(Complex z1) {
  return Complex{log(z1.r), z1.phi};
}

friend Complex pow(Complex z1, Complex z2) {
  return exp(z2 * log(z1));
}
```

Überladen Sie den Shiftoperator << geeignet, sodass die Ausgabe komplexer Zahlen wie in der Standardbibliothek erfolgt.

Beispiel. Für $z = 41 + 23{,}1 \cdot i$ soll `(41,23.1)` ausgegeben[1] werden.

Es ist sinnvoll Methoden, die den Zustand des Objekts (also die Attribute) nicht verändern mit `const` zu markieren.

Ausgabe

```
double real() const {
  return r * cos(phi);
}

double imag() const {
  return r * sin(phi);
}

friend ostream& operator<<(ostream& stream, Complex z) {
  return stream << "(" << z.real() << "," << z.imag() << ")";
}
```

Erstellen Sie ein Hauptprogramm in dem zwei Zahlen z_1 und z_2 zunächst als `complex<double>`-Zahlen eingelesen werden. Erzeugen Sie auch die entsprechenden Zahlen vom Typ `Complex`. Berechnen und geben Sie die folgenden Ausdrücke für die Zahlen vom Typ `Complex` und zum Vergleich auch für die Zahlen vom Standarddatentyp `complex<double>` aus:

$$\frac{z_1 - z_2}{z_1 + z_2} \qquad \sqrt{z_1 \cdot z_2} \qquad \exp(z_1) \qquad \log(z_2) \qquad z_1^{z_2}$$

Hauptprogramm

```
int main()
{
  complex<double> w1,w2;

  cout << "z1 z2: ";
  cin >> w1 >> w2;

  Complex z1{w1.real(), w1.imag()}, z2{w2.real(), w2.imag()};

  cout << fixed << setprecision(numeric_limits<double>::digits10);
  cout << "(z1-z2)/(z1+z2):" << endl
       << "  " << (z1-z2)/(z1+z2) << " " << (w1-w2)/(w1+w2) << endl;
  cout << "sqrt(z1*z2):" << endl
       << "  " << sqrt(z1*z2) << " " << sqrt(w1*w2) << endl;
  cout << "exp(z1):" << endl
       << "  " << exp(z1) << " " << exp(w1) << endl;
  cout << "log(z2):" << endl
       << "  " << log(z2) << " " << log(w2) << endl;
  cout << "pow(z1,z2):" << endl
       << "  " << pow(z1,z2) << " " << pow(w1,w2) << endl;

  return 0;
}
```

[1] ohne aktive Ausgabe-Manipulatoren

complex.cpp

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

using namespace std;
```

Klasse

Hauptprogramm

Führen Sie Ihr Programm aus für $z_1 = 1 + i$ und $z_2 = 2 + i$.

```
z1 z2: (1, 1) (2, 1)
(z1-z2)/(z1+z2):
  (-0.230769230769231,0.153846153846154) (-0.230769230769231,0.153846153846154)
sqrt(z1*z2):
  (1.442615274452683,1.039778260055571) (1.442615274452683,1.039778260055571)
exp(z1):
  (1.468693939915886,2.287355287178843) (1.468693939915885,2.287355287178842)
log(z2):
  (0.804718956217050,0.463647609000806) (0.804718956217050,0.463647609000806)
pow(z1,z2):
  (-0.309743504928493,0.857658012588736) (-0.309743504928494,0.857658012588736)
```