Ludwig-Maximilians-Universität München
Mathematisches Institut
Gregor Kleen

Programmieren II für Studierende der Mathematik

Blatt 9 – Lösungsvorschlag

Aufgabe 10 Vereinbaren Sie ein Template für eine Funktion `pNorm`. Templateparameter sollen ein Gleitpunktzahlen-Typ T (diese Eigenschaft muss hier und im Folgenden nicht überprüft werden) und eine positive ganze Zahl p sein. `pNorm` soll für einen Parameter vom Datentyp `complex` aus der Standardlibrary (für den gegebenen Gleitpunktzahlen-Typ) die p-Norm dieser komplexen Zahl berechnen und als Wert des gegebenen Gleitpunktzahlen-Typs zurückgeben.

Hinweis. Die p-Norm $\|\cdot\|_p$ einer komplexen Zahl ist definiert als:

$$\|a + ib\|_p := \left(|a|^p + |b|^p\right)^{\frac{1}{p}}$$

pNorm

```
template<class T = double, unsigned int p = 2>
  T pNorm(const complex<T>& z) {
    return pow(pow(abs(z.real()), p) + pow(abs(z.imag()), p), static_cast<T>(1) / p);
}
```

Vereinbaren und implementieren Sie eine von `complex` abgeleitete Klasse `complex_norm`. Die Definition soll ein Template mit Parametern für den zu verwendenen Gleitpunktzahlentyp und eine Norm-Funktion sein. Es soll sich eine Instanz Ihres templates `pNorm` für den zweiten Parameter einsetzen lassen.

Die Klasse `complex_norm` soll eine konstante Methode `norm` haben, die die als template Parameter übergeben Norm-Funktion aufruft.

complex norm

```
template<class T = double, T _norm(const complex<T>&) = pNorm<T>>
  class complex_norm : public complex<T> {
    public:
      complex_norm(T real_, T imag_): complex<T>(real_, imag_) {}

      T norm() const {
        return _norm(*this);
      }
};
```

Vereinbaren Sie unter Verwendung von `complex_norm` und `pNorm` einen parametrisierten Typalias `complex_pnorm`. Parameter sollen ein Gleitpunktzahlentyp T und eine positive ganze Zahl p sein.

complex pnorm

```
template<class T = double, unsigned int p = 2>
  using complex_pnorm = complex_norm<T, pNorm<T, p>>;
```

Die p-Normen sind *äquivalent*. D.h. für jedes Paar $1 < p_1 \leq p_2 \in \mathbb{N} \cup \{\infty\}$ existiert eine Konstante $c \in \mathbb{R}$ sodass für alle $x \in \mathbb{C}$ gilt:

$$\|x\|_{p_1} \leq c\|x\|_{p_2}$$

Es gilt:

$$c = \frac{\|1+i\|_{p_1}}{\|1+i\|_{p_2}}$$

Implementieren Sie ein Hauptprogramm, das zunächst c für $p_1 = 2$ und $p_2 = 3$ berechnet und mit einer gut an die Genauigkeit angepassten Anzahl von Nachkommastellen ausgibt. Danach sollen solange komplexe Zahlen z_k von der Standardeingabe eingelesen werden, solange dies nicht fehlschlägt und jeweils das Verhältnis $c_k = \frac{\|z\|_2}{\|z\|_3}$ berechnet und ebenfalls ausgegeben werden. Es soll zudem ausgegeben werden ob $c_k \leq c$ gilt.

pnorms.cpp

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

using namespace std;

pNorm

complex norm

complex pnorm

int main() {
  complex_pnorm<double, 2> z1{1, 1};
  complex_pnorm<double, 3> z2{1, 1};

  double bound = z1.norm() / z2.norm();

  cout << setprecision(numeric_limits<double>::digits10 + 1) << boolalpha;
  cout << bound << endl;

  double real_, imag_;
  while (true) {
    cout << "real, imag = ";
    if (!(cin >> real_ >> imag_)) break;

    z1 = complex_pnorm<double, 2>{real_, imag_};
    z2 = complex_pnorm<double, 3>{real_, imag_};
    double val = z1.norm() / z2.norm();
    cout << val << " " << (val <= bound) << endl;
  }
}
```

Beispiel (Kontrollergebnis). Ihr Programm könnte sich verhalten, wie folgt:

```
1.122462048309373
real, imag = 1.5 2.0
```

```
1.111619628853093 true
real, imag = 7.0 7.0
1.122462048309373 true
real, imag = 3.7 42.1
1.003627499490993 true
real, imag =
```