

Programmieren II für Studierende der Mathematik

Blatt 11 – Lösungsvorschlag

Aufgabe 12 Ziel ist es die Lösung der vorhergegangenen Übungsaufgabe von der Äquivalenz von p -Normen in eine bessere Code-Struktur zu bringen und mit unittests zu versehen. Falls Sie das entsprechende Übungsblatt nicht (vollständig) bearbeitet haben, können Sie auch den bereitgestellten Lösungsvorschlag verwenden. Bei der Umstrukturierung von bestehendem Code ohne (wesentliche) Änderung der Funktionalität spricht man von *refactoring*.

Legen Sie ein Verzeichnis für Ihr Projekt an. Verschieben Sie Ihre Lösung in eine Quelldatei (z.B. und im Folgenden `pnorms.cpp`) unterhalb des von Ihnen erstellten Verzeichnisses. Formulieren Sie eine Datei `meson.build` um die Übersetzungseinheit `pnorms.cpp` zu einer ausführbaren Binärdatei zu übersetzen und legen Sie diese ebenfalls in dem Verzeichnis ab.

```
project('pnorms', 'cpp')

executable('pnorms', 'pnorms.cpp')
```

Sie sollten nun in der Lage sein mit einem geeigneten meson-Befehl das `builddir` anlegen zu lassen, ihre Lösung mit `ninja` zu übersetzen und Sie danach interaktiv auszuführen.

Erstellen Sie eine Datei `pnorms.h` mit Deklarationen für die in `pnorms.cpp` definierten templates. Inkludieren Sie `pnorms.h` auch in `pnorms.cpp`.

Hinweis. Sie werden die Definitionen, die Teil der Klasse `complex_norm` sind, in `pnorms.cpp` ändern müssen. Sie dürfen die Klasse `complex_norm` nicht erneut vereinbaren und müssen stattdessen ihre Komponenten außerhalb einer Klassen-Deklaration definieren.

pnorms_unittests/pnorms.h

```
#pragma once

#include <complex>

template<class T = double, unsigned int p = 2>
T pNorm(const std::complex<T>& z);

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

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

Deklaration pnorm bound

Definition complex norm

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

template<class T, T _norm(const complex<T>&)>
    T complex_norm<T, _norm>::norm() const {
        return _norm(*this);
    }
```

An dieser Stelle sollten Sie erneut in der Lage sein ihre Lösung zu übersetzen und interaktiv auszuführen.

Deklariert Sie in der Header-Datei und definieren Sie in `pnorms.cpp` eine zusätzliche Funktionstemplate `pnorm_bound`. Die Funktionen sollen einen Parameter z vom Typ `complex<T>` akzeptieren und einen Wert vom Typ T zurückgeben. Zusätzlich zu T soll das template zwei weitere Parameter p_1 und p_2 akzeptieren. Der zurückgegebene Wert soll $\frac{\|z\|_{p_1}}{\|z\|_{p_2}}$ sein.

Deklaration pnorm bound

```
template<class T = double, unsigned int p1, unsigned int p2>
    T pnorm_bound(std::complex<T> = std::complex<T>{1.0, 1.0});
```

Definition pnorm bound

```
template<class T, unsigned int p1, unsigned int p2>
    T pnorm_bound(complex<T> z) {
        complex_pnorm<T, p1> z1{z.real(), z.imag()};
        complex_pnorm<T, p2> z2{z.real(), z.imag()};
        return z1.norm() / z2.norm();
    }
```

Passen Sie Ihr Hauptprogramm in `pnorms.cpp` an, sodass dieses `pnorm_bound` geeignet verwendet, wo immer sinnvoll möglich.

Lagern Sie ihr Hauptprogramm aus in eine zweite Übersetzungseinheit `pnorms_interact.cpp`.

Hinweis. Es wird notwendig sein die in `pnorms_interact.cpp` verwendeten Instanzen der templates aus `pnorms.h` in `pnorms.cpp` explizit zu instanziiieren.

Sie werden zudem Ihre `meson.build` anpassen müssen, sodass `pnorms.cpp` nun als Programmbibliothek übersetzt und mit dem Resultat der Übersetzung von `pnorms_interact.cpp` gelinkt wird.

pnorms_unittests/pnorms.cpp

```
#include <complex>
#include <cmath>

#include "pnorms.h"

using namespace std;

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

Definition complex norm

Definition pnorm bound

```
template double pNorm<double, 2>(const complex<double>&);
template double pNorm<double, 3>(const complex<double>&);
template class complex_norm<double, pNorm<double, 2>>;
template class complex_norm<double, pNorm<double, 3>>;
template double pnorm_bound<double, 2, 3>(complex<double>);
```

pnorms_unittests/pnorms_interact.cpp

```
#include <iostream>
#include <iomanip>
#include <limits>

#include "pnorms.h"

using namespace std;

int main() {
    double bound = pnorm_bound<double, 2, 3>();

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

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

        double val = pnorm_bound<double, 2, 3>(complex<double>{real_, imag_});
        cout << val << " " << (val <= bound) << endl;
    }
}
```

pnorms_unittests/meson.build

```
project('pnorms', 'cpp')

pnorms_lib = library('libpnorms', 'pnorms.cpp')
executable('pnorms', 'pnorms_interact.cpp', link_with : pnorms_lib)
```

Meson Test

An dieser Stelle sollten Sie erneut in der Lage sein ihre Lösung zu übersetzen und interaktiv auszuführen.

Befehle

Befehle subproject setup

```
meson setup --wipe build
ninja -C build
```

Befehle Test

Fügen Sie Ihrem meson Projekt das unit testing framework Google Test als subproject hinzu. Lassen Sie es von meson aus der WrapDB installieren.

Befehle subproject setup

```
mkdir -p subprojects
meson wrap install gtest
```

Fügen Sie eine weitere Übersetzungseinheit `pnorms_unittests.cpp` hinzu. Spezifizieren Sie in `meson.build` geeignet, dass das Ergebnis der Übersetzung von `pnorms_unittests.cpp` zu einer ausführbaren Binärdatei mit dem Befehl `meson test` ausführbar sein soll. Geben Sie hierbei auch an, dass die Binärdatei gelinkt werden soll mit der in der meson Variable `gtest_main_dep` im subproject `gtest` bereitgestellten dependency.

Hinweis. Es ist dann nicht notwendig in `pnorms_unittests.cpp` ein Hauptprogramm zu implementieren.

Meson Test

```
gtest = subproject('gtest')
test('gtest',
    executable('pnorms_gtest', 'pnorms_unittests.cpp',
        link_with : pnorms_lib,
        dependencies : gtest.get_variable('gtest_main_dep')
    )
)
```

Selbst mit `pnorms_unittests.cpp` einer komplett leeren Datei sollten Sie Ihr Projekt weiterhin übersetzen und interaktiv ausführen können. Es sollte auch möglich sein mit einem geeigneten `meson test`-Befehl die aus `pnorms_unittests.cpp` übersetzte ausführbare Binärdatei ausführen zu lassen. Es werden hierbei aber natürlich noch keinerlei Tests ausgeführt.

Befehle Test

```
meson test -C build
```

Implementieren Sie in `pnorms_unittests.cpp` beliebig viele unit tests, die Ihnen sinnvoll erscheinen.

Es sei im Folgenden $B(z) := \frac{\|z\|_2}{\|z\|_3}$; Implementieren Sie aber mindestens auch eine Test-Suite `PNorms` bestehend aus den folgenden Tests:

- $B(1 + i) \approx 1,122\,462\,048\,309$

Hinweis. Verwenden Sie das von Google Test bereitgestellte Präprozessor-Makro `EXPECT_NEAR(a, a', ε)`,

welches prüft, dass gilt: $|a - a'| \leq \varepsilon$.

ε soll hierbei nicht unnötig groß sein.

- $B(1,5 + 2,0i) \leq B(1 + i)$

Hinweis. Verwenden Sie hier und im Folgenden das von Google Test bereitgestellte Präprozessor-Makro `EXPECT_LE` (analog zu `EXPECT_EQ`).

- $B(7,0 + 7,0i) \leq B(1 + i)$
- $B(3,7 + 42,1i) \leq B(1 + i)$

pnorms_unittests/pnorms_unittests.cpp

```
#include <gtest/gtest.h>
#include <complex>

#include "pnorms.h"

namespace {
    using namespace std;

    const double bound = pnorm_bound<double, 2, 3>();

    TEST(PNorms, ExpectedBound) {
        const double expected_bound = 1.122462048309373;
        EXPECT_NEAR(bound, expected_bound, 1e-12);
    }
    TEST(PNorms, Example1) {
        EXPECT_LE((pnorm_bound<double, 2, 3>(complex<double>{1.5, 2.0})), bound);
    }
    TEST(PNorms, Example2) {
        EXPECT_LE((pnorm_bound<double, 2, 3>(complex<double>{7.0, 7.0})), bound);
    }
    TEST(PNorms, Example3) {
        EXPECT_LE((pnorm_bound<double, 2, 3>(complex<double>{3.7, 42.1})), bound);
    }
}
```

```
The Meson build system
Version: 1.1.0
Source dir: ../pnorms_unittests
Build dir: ../pnorms_unittests/build
Build type: native build
Project name: pnorms
Project version: undefined
C++ compiler for the host machine: g++ (gcc 12.2.0 "g++ (GCC) 12.2.0")
C++ linker for the host machine: g++ ld.bfd 2.40
Host machine cpu family: x86_64
Host machine cpu: x86_64

Executing subproject gtest
```

```
gtest| Project name: gtest
gtest| Project version: 1.14.0
gtest| C++ compiler for the host machine: g++ (gcc 12.2.0 "g++ (GCC) 12.2.0")
gtest| C++ linker for the host machine: g++ ld.bfd 2.40
gtest| Run-time dependency threads found: YES
gtest| Dependency threads found: YES unknown (cached)
gtest| Dependency threads found: YES unknown (cached)
gtest| Dependency threads found: YES unknown (cached)
gtest| Build targets in project: 2
gtest| Subproject gtest finished.

Build targets in project: 3

pnorms undefined

Subprojects
  gtest: YES

Found ninja-1.11.1 at ../bin/ninja
ninja: Entering directory `build'
[1/9] Compiling C++ object liblibpnorms.so.p/pnorms.cpp.o
[2/9] Compiling C++ object pnorms.p/pnorms_interact.cpp.o
[3/9] Linking target liblibpnorms.so
[4/9] Generating symbol file liblibpnorms.so.p/liblibpnorms.so.symbols
[5/9] Linking target pnorms
[6/9] Compiling C++ object
↳ pnorms_gtest.p/subprojects_googletest-1.14.0_googletest_src_gtest_main.cc.o
[7/9] Compiling C++ object ../pnorms_unittests.cpp.o
[8/9] Compiling C++ object
↳ pnorms_gtest.p/subprojects_googletest-1.14.0_googletest_src_gtest-all.cc.o
[9/9] Linking target pnorms_gtest
ninja: no work to do.
ninja: Entering directory ../pnorms_unittests/build'
ninja: no work to do.
1/1 gtest OK          0.01s

Ok:                1
Expected Fail:     0
Fail:              0
Unexpected Pass:   0
Skipped:          0
Timeout:          0

Full log written to ../pnorms_unittests/build/meson-logs/testlog.txt
```