# Eine Einführung in modernes C++

## Teil 3 – Unittest, Tools und best practices

Paul Nykiel

27. April 2020

# Unittests

# Was sind Unittests?

- Möglichst kleine Komponenten des Codes testen (z.B. einzelne Funktion/Klasse)

# Was sind Unittests?

- Möglichst kleine Komponenten des Codes testen (z.B. einzelne Funktion/Klasse)
- Nicht komplette Codebase notwendig

# Was sind Unittests?

- Möglichst kleine Komponenten des Codes testen (z.B. einzelne Funktion/Klasse)
- Nicht komplette Codebase notwendig
- Restlicher Code kann auf Korrektheit vertrauen

# Was sind Unittests?

- Möglichst kleine Komponenten des Codes testen (z.B. einzelne Funktion/Klasse)
- Nicht komplette Codebase notwendig
- Restlicher Code kann auf Korrektheit vertrauen
- Vorgehen: Festdefinierte Eingabe und daraus resultierende Ausgabe

# Google Test - Einführung

- Tests von einem Modul werden in einer Testsuite zusammengefasst

# Google Test - Einführung

- Tests von einem Modul werden in einer Testsuite zusammengefasst
- Innerhalb einer Testsuite können beliebig viele Tests angelegt werden

# Google Test - Einführung

- Tests von einem Modul werden in einer Testsuite zusammengefasst
- Innerhalb einer Testsuite können beliebig viele Tests angelegt werden
- Jeder Test sollte genau eine Funktionalität prüfen

# Google Test - Einführung

- Tests von einem Modul werden in einer Testsuite zusammengefasst
- Innerhalb einer Testsuite können beliebig viele Tests angelegt werden
- Jeder Test sollte genau eine Funktionalität prüfen
- Bedingungen mit Makros testen

# Google Test - Funktionen

```
TEST(Sqrt, Simple) {
    EXPECT_DOUBLE(sqrt(4), 2);
}

TEST(Sqrt, NonInteger) {
    EXPECT_DOUBLE_EQ(sqrt(17), 4.12310562562);
}

TEST(Sqrt, Negative) {
    EXPECT_THROW(sqrt(-1), std::runtime_error);
}
```

# Google Test - Klassen

```
TEST(Averager, Single) {
    Averager avg;
    avg.submit(1);
    avg.submit(3);

    EXPECT_DOUBLE_EQ(avg.getAverage(), 2);
}

TEST(Averager, ErrorNone) {
    Averager avg;
    EXPECT_THROW(avg.getAverage(), std::runtime_error);
}
```

# Tools

# Demo: Debugger

# Demo: Valgrind

# Demo: clang-tidy

# Best Practices

# Allgemein

- Auskommentierter Code entfernen

# Allgemein

- Auskommentierter Code entfernen
- Warnings beachten

# Allgemein

- Auskommentierter Code entfernen
- Warnings beachten
- Sichtbarkeit minimieren

# Allgemein

- Auskommentierter Code entfernen
- Warnings beachten
- Sichtbarkeit minimieren
- Duplicate Code vermeiden

# Allgemein

- Auskommentierter Code entfernen
- Warnings beachten
- Sichtbarkeit minimieren
- Duplicate Code vermeiden
- Einzeilige `if` klammern

# Allgemein

- Auskommentierter Code entfernen
- Warnings beachten
- Sichtbarkeit minimieren
- Duplicate Code vermeiden
- Einzeilige `if` klammern
- Magic Numbers extrahieren

# Funktionen - In Parameter

In-Parameter: Wird von der Funktion nur gelesen

# Funktionen - In Parameter

## In-Parameter: Wird von der Funktion nur gelesen

Für In-Parameter gilt:

- Wenn trivial zu kopieren: by-value: `double sqrt(double v);`

# Funktionen - In Parameter

## In-Parameter: Wird von der Funktion nur gelesen

Für In-Parameter gilt:

- Wenn trivial zu kopieren: by-value: `double sqrt(double v);`
- Für größere Objekte: als `const`-Referenz:
  `double getMax(const std::vector<double> &v);`

# Funktionen - In-Out Parameter

In-Out-Parameter: Wird von der Funktion gelesen und geschrieben

# Funktionen - In-Out Parameter

## In-Out-Parameter: Wird von der Funktion gelesen und geschrieben

Für In-Out-Parameter gilt:

- Als Referenz an Funktion übergeben:
  ```
  void removeDuplicates(std::vector<int> &v);
  ```

# Funktionen - Out Parameter

Out-Parameter: Wird von der Funktion nur geschrieben

# Funktionen - Out Parameter

## Out-Parameter: Wird von der Funktion nur geschrieben

Für Out-Parameter gilt:

- Immer via `return`

# Funktionen - Out Parameter

- Aber C kann nur einen Wert auf einmal returnen?

# Funktionen - Out Parameter

- Aber C kann nur einen Wert auf einmal returnen? `std::tuple`, `std::pair` oder `struct`

# Funktionen - Out Parameter

- Aber C kann nur einen Wert auf einmal returnen? `std::tuple`, `std::pair` oder `struct`
- `struct` für komplexere Datentypen

# Funktionen - Out Parameter

- Aber C kann nur einen Wert auf einmal returnen? `std::tuple`, `std::pair` oder `struct`
- `struct` für komplexere Datentypen
- Structure Binding: `auto [min, max, avg] = getMinMaxAvg(list);`

# Funktionen - Out Parameter

- Aber C kann nur einen Wert auf einmal returnen? `std::tuple`, `std::pair` oder `struct`
- `struct` für komplexere Datentypen
- Structure Binding: `auto [min, max, avg] = getMinMaxAvg(list);`
- Für Werte die in manchen Fällen nicht existieren: `std::optional`.

```
std::optional<int> res = calculateIfPossible();

if (res.has_value()) {
    std::cout << res.value() << std::endl;
}
```

# C-Konstrukte

- Referenzen bzw. Smart-Pointer statt Raw-Pointer

# C-Konstrukte

- Referenzen bzw. Smart-Pointer statt Raw-Pointer
- `enum class` statt `enum`

# C-Konstrukte

- Referenzen bzw. Smart-Pointer statt Raw-Pointer
- `enum class` statt `enum`
- `std::array` und `std::vector` statt C-Style Arrays (`int arr[10];`)

# C-Konstrukte

- Referenzen bzw. Smart-Pointer statt Raw-Pointer
- `enum class` statt `enum`
- `std::array` und `std::vector` statt C-Style Arrays (`int arr[10];`)
- (Template-) Funktionen statt Makros

# Header

- Header immer mit Include-Guards schützen

```
#ifndef MODULE_NAME_HEADER_NAME_HPP
#define MODULE_NAME_HEADER_NAME_HPP

int function1();
void function2();

class C {};

#endif
```

- Alle verwendeten Header inkludieren

# Header

- Header immer mit Include-Guards schützen

```
#ifndef MODULE_NAME_HEADER_NAME_HPP
#define MODULE_NAME_HEADER_NAME_HPP

int function1();
void function2();

class C {};

#endif
```

- Alle verwendeten Header inkludieren
- Code in einer .cpp Datei implementieren, nicht im Header

# Abschluss

# Was fehlt?

- R-Value Referenzen, forward/universal Referenzen

# Was fehlt?

- R-Value Referenzen, forward/universal Referenzen
- Move

# Was fehlt?

- R-Value Referenzen, forward/universal Referenzen
- Move
- Destruktor und Copy / Move Konstruktor

# Was fehlt?

- R-Value Referenzen, forward/universal Referenzen
- Move
- Destruktor und Copy / Move Konstruktor
- Operatorenüberladung

# Was fehlt?

- R-Value Referenzen, forward/universal Referenzen
- Move
- Destruktor und Copy / Move Konstruktor
- Operatorenüberladung
- Friend Definition

# Was fehlt?

- R-Value Referenzen, forward/universal Referenzen
- Move
- Destruktor und Copy / Move Konstruktor
- Operatorenüberladung
- Friend Definition
- Meta-Programming

# Mehr Informationen

- `en.cppreference.com`

# Mehr Informationen

- `en.cppreference.com`
- `github.com/isocpp/CppCoreGuidelines`

# Mehr Informationen

- `en.cppreference.com`
- `github.com/isocpp/CppCoreGuidelines`
- `godbolt.org`

# Mehr Informationen

- `en.cppreference.com`
- `github.com/isocpp/CppCoreGuidelines`
- `godbolt.org`
- `git.spatz.wtf/spatzenhirn/cppcmakeintro`

# Mehr Informationen

- `en.cppreference.com`
- `github.com/isocpp/CppCoreGuidelines`
- `godbolt.org`
- `git.spatz.wtf/spatzenhirn/cppcmakeintro`
- Scott Meyers: Effective Modern C++

# Praxis

# Praxis

- Schreibt Unittests für den Huffman Encoder

# Praxis

- Schreibt Unittests für den Huffman Encoder
- Untersucht das Verhalten: meldet Valgrind Probleme? meldet clang-tidy welche?

# Praxis

- Schreibt Unittests für den Huffman Encoder
- Untersucht das Verhalten: meldet Valgrind Probleme? meldet clang-tidy welche?
- Lest euch die Code-Richtlinien (`https://git.spatz.wtf/spatzenhirn/wiki/-/wikis/Software/Cup2021/CodeRichtlinien`) durch und überprüft ob euer Code die Richtlinien erfüllt.