University of Applied Sciences
HOCHSCHULE EMDEN·LEER

Fachbereich Technik
Abteilung Elektrotechnik und Informatik

PROGRAMMIEREN III
AUFGABE 4

Gruppe A1
Studiengang Elektrotechnik
Vorgelegt von

Yaman Alsaady Matr. Nr. 7023554
Oliver Schmidt Matr. Nr. 7023462

Emden, 7. Dezember 2023

Betreut von
Dr. Olaf Bergmann
Dipl.-Ing. Behrend Pupkes

Kapitel 1

Fragen

1.1 Aufgabe 4.6

Bewerten Sie die Ergebnisse aus den Aufgaben 4.4 und 4.5 vor dem Hintergrund der Implementierung des Funktors Data in stat.h. Was fällt auf?

Bei einer gleichmäßigen Verteilung ergibt sich die durchschnittliche Häufigkeit durch die folgende Formel: "FORMEL Schreiben".

$$\frac{n}{1 + max - min} - Average$$

Bei einer enormen Anzahl an zufälligen Zahlen sollte die durchschnittliche Häufigkeit bei der errechneten Zahl liegen und gleichmäßig auf alle Elemente verteilt. Wenn die Anzahl an zufälligen Zahlen kleiner ist, wird die Verteilung nicht mehr gleichmäßig sein.

Kapitel 2

Quellencode

2.1 Datei ’stats.cc’

Listing 2.1 stat.cc

```cpp
#include <algorithm>
#include <functional>
#include <iomanip>
#include <iostream>
#include <iterator>
#include <map>
#include <numeric>
#include <stdexcept>
#include <vector>

#include "stats.h"

using namespace std;
using namespace std::placeholders;

/**
 * @brief Hauptfunktion des Programms.
 *
 * - Fuellt einen Vektor mit Zufallszahlen.
 * - Sortiert den Vektor aufsteigend und gibt ihn aus.
 * - Ermittelt die Haeufigkeit der Zufallszahlen und stellt sie als Balkengrafik
 * dar.
 * - Berechnet den Durchschnitt der Haeufigkeiten.
 *
 * @return Rueckgabewert 0 bei erfolgreicher Ausfuehrung.
 */
int main() {
  int num = 3902365, min = 1, max = 100;
  vector<unsigned int> v;

  // 1. Fuellen von v mit 3902365 Zufallswerten.
  generate_n(back_inserter(v), num, Data(min, max));

  // 2. aufsteigend sortieren und mit ostream_iterator ausgeben.
  sort(v.begin(), v.end());
  // Ausgabe rausgenomme, weil das nervig ist.
  copy(v.begin(), v.end(), ostream_iterator<unsigned int>(cout, ", "));
  cout << endl;

  // 3. Haeufigkeit ermitteln.
  map<unsigned int, unsigned int> frequencyMap;
  for_each(v.begin(), v.end(),
           [&frequencyMap](unsigned int val) { frequencyMap[val]++; });

  // 4. Haeufigkeit als Balkengrafik darstellen.
  for (const auto &entry : frequencyMap) {
```

```
47      cout << setw(3) << entry.first << ": " << setw(6) << entry.second << " ";
48      for (unsigned int i = 0; i < entry.second / 1000; ++i) {
49        cout << "#";
50      }
51      cout << endl;
52    }
53
54    // 5. Durchschnitt berechnen und ausgeben.
55    double average = accumulate(frequencyMap.begin(), frequencyMap.end(), 0.0,
56                                [](double sum, const auto &entry) {
57                                  return sum + entry.second;
58                                }) /
59                     frequencyMap.size();
60    cout << "Durchschnittliche Haeufigkeit: " << average << endl;
61
62    double average_value = accumulate(begin(v), end(v), 0.0) / v.size();
63
64    cout << "Durchschnitt der Zufallszahlen: " << average_value << endl;
65
66    cout << "Abweichung der Durchschnittliche Haeufigkeit"<< endl;
67    cout << "(n/(max-min+1))-average = ";
68    std::cout << (double(num)/(max-min+1))-average << std::endl;
69
70    return 0;
71  }
```

2.2 Datei ’stats.h’

Listing 2.2 stat.h

```
#ifndef STATS_H_
#define STATS_H_
#include <random>

/**
 * @brief Klasse fuer die Generierung von Zufallszahlen im angegebenen Intervall.
 */
class Data {
  std::mt19937 rng;
  std::uniform_int_distribution<unsigned int> dist;

public:
  /**
   * @brief Konstruktor fuer die Initialisierung des Zufallszahlengenerators und
   * der Verteilung.
   * @param a Untere Grenze des Intervalls.
   * @param b Obere Grenze des Intervalls.
   */
  Data(unsigned int a, unsigned int b)
      : rng(std::random_device()()), dist(a, b) {}

  /**
   * @brief Operatorfunktion zum Generieren einer Zufallszahl.
   * @return Zufallszahl im spezifizierten Intervall.
   */
  auto operator()(void) { return dist(rng); }
};

#endif /* STATS_H_ */
```

2.3 Datei ’Makefile’

Listing 2.3 Makefile

```
CXX = g++
CFLAGS = -Wall -Wextra -pedantic
SRC1 = $(wildcard *.cc)
SRC2 = $(wildcard *.cpp)
OBJ1 = $(patsubst %.cc, build/%.o, $(SRC1))
OBJ2 = $(patsubst %.cpp, build/%.o, $(SRC2))

build/main: $(OBJ1) $(OBJ2)
  $(CXX) $(CFLAGS) $(OBJ1) $(OBJ2) -o $@

build/%.o: %.cc
  @mkdir -p build
  ${CXX} ${CFLAGS} -c $< -o $@

build/%.o: %.cpp
  @mkdir -p build
  ${CXX} ${CFLAGS} -c $< -o $@

all: clean build/main

clean:
  rm -rf build

run:
  ./build/main
```

Kapitel 3

Klassen-Verzeichnis

3.1 Auflistung der Klassen

Hier folgt die Aufzählung aller Klassen, Strukturen, Varianten und Schnittstellen mit einer Kurzbeschreibung:

Kapitel 4

Datei-Verzeichnis

4.1 Auflistung der Dateien

Hier folgt die Aufzählung aller Dateien mit einer Kurzbeschreibung:

Kapitel 5

Klassen-Dokumentation

5.1 Data Klassenreferenz

Klasse fuer die Generierung von Zufallszahlen im angegebenen Intervall.

`#include <stats.h>`

Öffentliche Methoden

- Data (unsigned int a, unsigned int b)

  Konstruktor fuer die Initialisierung des Zufallszahlengenerators und der Verteilung.
- auto operator() (void)

  Operatorfunktion zum Generieren einer Zufallszahl.

Private Attribute

- std::mt19937 rng
- std::uniform_int_distribution< unsigned int > dist

5.1.1 Ausführliche Beschreibung

Klasse fuer die Generierung von Zufallszahlen im angegebenen Intervall.

5.1.2 Beschreibung der Konstruktoren und Destruktoren

5.1.2.1 Data()

```
Data::Data (
            unsigned int a,
            unsigned int b )  [inline]
```

Konstruktor fuer die Initialisierung des Zufallszahlengenerators und der Verteilung.

Parameter

a	Untere Grenze des Intervalls.
b	Obere Grenze des Intervalls.

5.1.3 Dokumentation der Elementfunktionen

5.1.3.1 operator()()

```
auto Data::operator() (
            void  )  [inline]
```

Operatorfunktion zum Generieren einer Zufallszahl.

Rückgabe

Zufallszahl im spezifizierten Intervall.

5.1.4 Dokumentation der Datenelemente

5.1.4.1 dist

```
std::uniform_int_distribution<unsigned int> Data::dist  [private]
```

5.1.4.2 rng

```
std::mt19937 Data::rng  [private]
```

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- stats.h

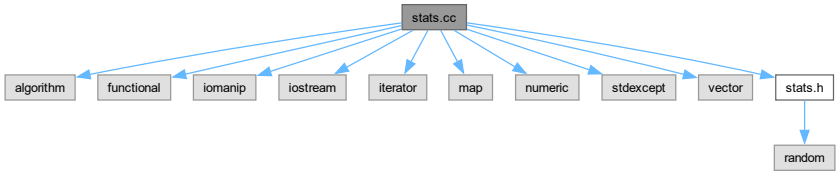
Kapitel 6

Datei-Dokumentation

6.1 stats.cc-Dateireferenz

```
#include <algorithm>
#include <functional>
#include <iomanip>
#include <iostream>
#include <iterator>
#include <map>
#include <numeric>
#include <stdexcept>
#include <vector>
#include "stats.h"
```

Include-Abhängigkeitsdiagramm für stats.cc:



Funktionen

- int main ()

  Hauptfunktion des Programms.

6.1.1 Dokumentation der Funktionen

6.1.1.1 main()

```
int main ( )
```

Hauptfunktion des Programms.

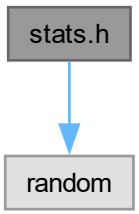
- Fuellt einen Vektor mit Zufallszahlen.
- Sortiert den Vektor aufsteigend und gibt ihn aus.
- Ermittelt die Haeufigkeit der Zufallszahlen und stellt sie als Balkengrafik dar.
- Berechnet den Durchschnitt der Haeufigkeiten.

Rückgabe

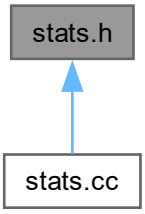
Rueckgabewert 0 bei erfolgreicher Ausfuehrung.

6.2 stats.h-Dateireferenz

`#include <random>`
Include-Abhängigkeitsdiagramm für stats.h:



Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:



Klassen

- class Data

  Klasse fuer die Generierung von Zufallszahlen im angegebenen Intervall.

6.3 stats.h

gehe zur Dokumentation dieser Datei

```
00001 #ifndef STATS_H_
00002 #define STATS_H_
00003 #include <random>
00004
00008 class Data {
00009   std::mt19937 rng;
00010   std::uniform_int_distribution<unsigned int> dist;
00011
00012 public:
00019   Data(unsigned int a, unsigned int b)
00020       : rng(std::random_device()()), dist(a, b) {}
00021
00026   auto operator()(void) { return dist(rng); }
00027 };
00028
00029 #endif /* STATS_H_ */
```

Index