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, 4. Dezember 2023

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

Kapitel 1

Quellencode

1.1 Datei ’stats.cc’

Listing 1.1 stat.cc

```
#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.
 *
 * - F llt einen Vektor mit Zufallszahlen.
 * - Sortiert den Vektor aufsteigend und gibt ihn aus.
 * - Ermittelt die H ufigkeit der Zufallszahlen und stellt sie als Balkengrafik
 * dar.
 * - Berechnet den Durchschnitt der H ufigkeiten.
 *
 * @return R ckgabewert 0 bei erfolgreicher Ausf hrung.
 */
int main() {
  vector<unsigned int> v;

  // 1. F llen von v mit 3902365 Zufallswerten.
  generate_n(back_inserter(v), 3902365, Data(1, 100));

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

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

  // 4. H ufigkeit als Balkengrafik darstellen.
  for (const auto &entry : frequencyMap) {
    cout << setw(3) << entry.first << ": " << setw(6) << entry.second << " ";
    for (unsigned int i = 0; i < entry.second / 1000; ++i) {
```

```
47        cout << "#";
48      }
49      cout << endl;
50    }
51
52    // 5. Durchschnitt berechnen und ausgeben.
53    double average = accumulate(frequencyMap.begin(), frequencyMap.end(), 0.0,
54                                [](double sum, const auto &entry) {
55                                  return sum + entry.second;
56                                }) /
57                     frequencyMap.size();
58    cout << "Durchschnittliche H ufigkeit: " << average << endl;
59
60    double average_value = accumulate(begin(v), end(v), 0.0) / v.size();
61
62    cout << "Durchschnitt der Zufallszahlen: " << average_value << endl;
63
64    return 0;
65  }
```

1.2 Datei ’stats.h’

Listing 1.2 stat.h

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

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

public:
  /**
   * @brief Konstruktor f r 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_ */
```

1.3 Datei ’Makefile’

Listing 1.3 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 2

Klassen-Verzeichnis

2.1 Auflistung der Klassen

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

Kapitel 3

Datei-Verzeichnis

3.1 Auflistung der Dateien

Hier folgt die Aufzählung aller dokumentierten Dateien mit einer Kurzbeschreibung:

Kapitel 4

Klassen-Dokumentation

4.1 Data Klassenreferenz

Klasse für die Generierung von Zufallszahlen im angegebenen Intervall.

`#include <stats.h>`

Öffentliche Methoden

- Data (unsigned int a, unsigned int b)

  Konstruktor für 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**

4.1.1 Ausführliche Beschreibung

Klasse für die Generierung von Zufallszahlen im angegebenen Intervall.

4.1.2 Beschreibung der Konstruktoren und Destruktoren

4.1.2.1 Data()

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

Konstruktor für die Initialisierung des Zufallszahlengenerators und der Verteilung.

Parameter

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

4.1.3 Dokumentation der Elementfunktionen

4.1.3.1 operator()()

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

Operatorfunktion zum Generieren einer Zufallszahl.

Rückgabe

Zufallszahl im spezifizierten Intervall.

Die Dokumentation für diese Klasse wurde erzeugt aufgrund der Datei:

- /home/yaman/Studium/3.Semster/Programmieren_3/Aufgaben/Aufgabe_4_Olli/src/stats.h

Kapitel 5

Datei-Dokumentation

5.1 stats.h

```
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