BERGISCHE UNIVERSITÄT
GESAMTHOCHSCHULE WUPPERTAL
GAUSS-STRASSE 20
42097 WUPPERTAL
(Korrespondenzanschrift)
42119 WUPPERTAL
(Lieferanschrift)
TELEX 8 592 262 bughw
TELEFAX (0202) 439-2901
TELEFON (0202) 439-1

Fachbereich 7
MATHEMATIK

Prof. Dr. Hans-Jürgen Buhl
Praktische Informatik / Numerik

e-mail: Juergen.Buhl@math.uni-wuppertal.de

Informatik I

WS98/99 – Übungsblatt 5

Bearbeitungstermin: 49. Kalenderwoche

Aufgabe 1. *const-Parameter bei Funktionen*

Ändern Sie die Signatur der Funktion `average`

```
double average(double v[], int dim)
{
  assert(dim > 0);
  double result(0);

  for (int i = 0; i < dim; i ++)
    result += v[i];

  return result/dim;
};
```

aus Aufgabe 4/Übungsblatt 3 ab zu

```
double average(const double v[], const int dim)
```

und testen Sie zunächst, ob die alte Funktionalität erhalten bleibt. Versuchen Sie danach das Verhalten Ihres C++ Entwicklungssystems zu bestimmen, wenn im ausführbaren Teil von `average` nun „versehentlich" eine Wertzuweisung an `v` bzw. an `dim` durchgeführt wird. Archivieren Sie dieses Verhalten in Ihrer Kladde!

Ändern Sie Ihre Funktion `geomMittelwert` (Aufgabe 4/Übungsblatt 4) analog ab, und testen Sie.

Aufgabe 2. *Mittelwert und Standardabweichung*

Für sehr lange Datensätze $\{x_i\}_{i=1}^n$ soll nun der *Mittelwert*

$$\mu := \left(\sum_{i=1}^n x_i\right) / \ n$$

und die Standardabweichung

$$\sigma := \sqrt{\frac{\sum_{i=1}^n x_i^2 - \frac{\left(\sum_{i=1}^n x_i\right)^2}{n}}{n-1}}$$

bestimmt werden.

Dazu eignet sich die Übergabe des Datensatzes als Vektor nur sehr eingeschränkt. Eine andere Lösung wird im folgenden Programm vorgestellt:

```
//////////////////////////////////
// Datei:   Mittelwert.cc
// Version: 1.0
// Zweck:   Klasse Statistik fuer Mittelwert/Standardabweichung
// Datum:   24.11.98
//////////////////////////////////

#include <iostream>
#include <math.h>
#include <assert.h>

//using namespace std;

class Statistik{

double SumX;
double SumX2;
int n; // Anzahl der Datenelemente

public:
Statistik(): SumX(0.0), SumX2(0.0), n(0) {};

void restart()
{
SumX = 0.0; SumX2 = 0.0; n = 0;
};

void AddItem(double value)
{
n++;
SumX  += value;
SumX2 += pow(value, 2);
};

void DeleteItem(double value)
{
assert(n>0);
n--;
SumX  -= value;
SumX2 -= pow(value, 2);
};

double MeanValue() const
{
assert(n>0);
```

```
return SumX / n;
};

double StdDeviation() const
{
assert(n>1);
return sqrt( (SumX2 - pow(SumX, 2) / n) / ( n - 1 ) );
};

};

int main()
{
  Statistik Daten1;
  Daten1.AddItem(1.0);
  Daten1.AddItem(1.1);
  Daten1.AddItem(1.0);
  Daten1.AddItem(1.0);
  cout << "Mittelwert Datensatz 1: " << Daten1.MeanValue() << endl;
  cout << "Standardabweichung Datensatz 1: " << Daten1.StdDeviation() << endl;

  Statistik Daten2;
  Daten2.AddItem(1.0);
  Daten2.AddItem(2.0);
  cout << "Mittelwert Datensatz 2: " << Daten2.MeanValue() << endl;
  cout << "Standardabweichung Datensatz 2: " << Daten2.StdDeviation() << endl;

  Daten1.AddItem(0.9);
  cout << "Mittelwert Datensatz1: " << Daten1.MeanValue() << endl;
  cout << "Standardabweichung Datensatz1: " << Daten1.StdDeviation() << endl;

  Daten1.restart();
  Daten1.AddItem(1.0);
  Daten1.AddItem(1.0);
  cout << "Mittelwert Datensatz 1 (neu): " << Daten1.MeanValue() << endl;
  cout << "Standardabweichung Datensatz 1 (neu): " << Daten1.StdDeviation()
       << endl;

  return 0;
}
```

Bringen Sie das Programm zum Ablauf und testen Sie! Diskutieren Sie die Vor- und Nachteile dieser Version.

Aufgabe 3. *Güte des Pseudozufallszahlengenerators* `rand()`

Benutzen Sie die Klasse `Statistik` aus Aufgabe 2, um die Güte des Pseudozufallszahlengenerators `rand()` (vgl. Aufgabe 3/Übungsblatt 4) für verschiedene Initialwerte und für verschieden lange Datensätze zu bestimmen.

Aufgabe 4. *Fortsetzung Pseudozufallszahlen*

Einen besseren Pseudozufallszahlengenerator erhalten Sie mittels:

```
//////////////////////////////////
// Datei:   random2.cc
// Version: 1.1
// Zweck:   Pseudo-Zufallszahlen
// Autor:   Hans-Juergen Buhl
// Datum:   17.09.1998
//////////////////////////////////

#include <iostream>
#include <cstdlib>
#include <time.h>

//using namespace std;

int main()
{

  time_t walltime(time(0));
  cout << walltime << endl << endl;

  srand48(walltime);

  for (int i=0; i<=20; i++)
     cout << drand48() << endl;

  cout << endl << "Pseudozufallszahlen in [0.0, 1.0)" << endl;

  return 0;
}
```

Informieren Sie sich über die Pseudozufallszahlengeneratoren `drand48()`, `lrand48()` und `mrand48()` sowie die Initialisierungsroutine `srand48(long)` mittels `man srand48`.

Testen Sie obiges Programm. Wiederholen Sie die Gütebestimmungen aus Aufgabe 3 für die drei Pseudozufallszahlengeneratoren `drand48()`, `lrand48()` und `mrand48()`.