Prof. Dr. Carsten Meyer
Fachbereich Informatik & Elektrotechnik
Fachhochschule Kiel

Programmieren in C++ – Sommersemester 2015

Aufgabenblatt 1

Abgabe: Donnerstag, 23.04.2015, 11:30

Aufgabe 1 (Kontrollstrukturen, Zufallszahlen):

Schreiben Sie ein Programm, das Zufallszahlen zwischen 0 und 1 generiert, und zwar so lange, bis die generierte Zufallszahl größer als 0.85 ist. Die Zufallszahlen sollen auf dem Bildschirm ausgegeben werden. Weiterhin sollen von den generierten Zufallszahlen Summe, Mittelwert, Minimum und Maximum berechnet und auf dem Bildschirm ausgegeben werden (zusammen mit der Anzahl der gezogenen Zufallszahlen). Das Programm soll insgesamt 5 Durchläufe generieren.

Die Lösung der Aufgabe soll sowohl die Implementierung als auch eine Bildschirmausgabe des kompletten Programms (5 Durchläufe) umfassen.

Hinweise:

1.) Zur (Pseudo)Zufallszahlenberechnung verwenden Sie die Funktion **`rand()`**, die eine Integralzahl zwischen 0 und RAND_MAX zurückgibt. Diese (Pseudo-) Zufallszahlenberechnung muß initialisiert werden, damit nicht bei jedem erneuten Programmaufruf dieselbe Sequenz von Zufallszahlen generiert wird. Hierzu rufen Sie einmalig gleich zu Beginn der **`main()`**-Funktion die folgende Funktion auf:

```
srand( time(NULL) );
```

Achtung: Unter Linux muß für **`srand`** ggf. die **`cstdlib`** eingebunden werden!

2.) Zur Ausgabe der Zahlen verwenden Sie entweder die C-Funktion "**`printf`**" oder die C++ Ausgabe über "**`cout`**".
Beispiel: Eine Gleitpunktzahl z kann ebenso wie eine Integer-Zahl i und ein String s ausgegeben werden mit

```
cout << z << endl;
cout << i << endl;
cout << s << endl;
cout << "Hallo" << endl;
```

Das "**`endl`**" sorgt dabei für einen Zeilenumbruch (mehr später in der Vorlesung)

3.) Falls Sie Typumwandlungen benötigen, benutzen Sie bitte die Typumwandlungsoperatoren von C++!

Lösung:

```
#include <iostream>
#include <ctime> // für time
#include <cstdlib> // für srand, RAND_MAX

using namespace std;

int main (int argc, char* argv[] )
{
    int runs = 5; // 5 Durchlaeufe
    float schwelle = 0.85;

    srand( time(NULL) ); // Initialisierung Zufallszahlengenerator

    for ( int i = 0; i < runs; i++ )
    {
      cout << "Durchlauf " << i + 1 << endl;
      float zahl = 0.0; // Initialisierung Zufallszahl
      float sum = 0.0, min = 1.0, max = 0.0, mean = 0.0;
      int anzahl = 0;

      do
      {
          zahl = static_cast<float>( rand() ) / RAND_MAX;
                 // generiere Zufallszahl zwischen 0 und 1
                 // Division durch RAND_MAX, (C++-) Konvertierung nach float
          if ( zahl <= schwelle ) // gezogene Zahl könnte größer als Schwelle sein
          {
            cout << zahl << endl;
            anzahl++;            // aktualisiere Anzahl der gezogenen gültigen Zahlen
            sum += zahl;         // aktualisiere Summe
            if ( zahl < min )
              min = zahl;        // aktualisiere Minimum
            if ( zahl > max )
              max = zahl;        // aktualisiere Maximum
          }
      } while ( zahl <= schwelle );

      if ( anzahl > 0 )
      {
        mean = sum / anzahl;      // berechne Mittelwert
        cout << anzahl << " gueltige Zufallszahlen gezogen" << endl;
        cout << "Summe: " << sum << endl;
        cout << "Mittelwert: " << mean << endl;
        cout << "Minimum: " << min << endl;
        cout << "Maximum: " << max << endl;
        cout << endl;
      }
      else
      {
        cout << "Keine gueltige Zufallszahl gezogen!" << endl;
        cout << endl;
      }
    }
    return 0;
}
```

Programmausgabe:

```
Durchlauf 1
0.830409
0.0647908
0.424512
0.308451
0.811884
0.202612
0.727378
0.362407
0.428419
0.0196539
0.270241
0.769402
0.0480972
0.797082
0.554003
0.494003
16 gueltige Zufallszahlen gezogen
Summe: 7.11335
Mittelwert: 0.444584
Minimum: 0.0196539
Maximum: 0.830409

Durchlauf 2
0.127903
0.682791
0.0739463
0.246193
0.652608
0.282601
0.489059
0.148076
0.132359
0.406262
0.0717185
0.611408
12 gueltige Zufallszahlen gezogen
Summe: 3.92492
Mittelwert: 0.327077
Minimum: 0.0717185
Maximum: 0.682791

Durchlauf 3
0.33079
0.739402
0.138188
0.16126
0.653768
0.330607
0.818781
0.352367
0.594745
0.325938
0.294107
```

```
0.79458
12 gueltige Zufallszahlen gezogen
Summe: 5.53453
Mittelwert: 0.461211
Minimum: 0.138188
Maximum: 0.818781

Durchlauf 4
0.734642
0.220344
2 gueltige Zufallszahlen gezogen
Summe: 0.954985
Mittelwert: 0.477493
Minimum: 0.220344
Maximum: 0.734642

Durchlauf 5
0.312571
0.428907
0.391614
0.755486
0.84695
0.706442
0.176977
0.766137
0.0525834
0.362865
0.428388
0.226753
0.395062
0.0550554
0.515488
0.507584
0.826655
0.236915
0.643513
0.792199
20 gueltige Zufallszahlen gezogen
Summe: 9.42814
Mittelwert: 0.471407
Minimum: 0.0525834
Maximum: 0.84695

Press any key to continue . . .
```

Aufgabe 2 (Struktur, sizeof-Operator):

a) Mit dem **`sizeof`**-Operator kann die Anzahl an Bytes ermittelt werden, die Variablen eines bestimmten Datentyps im Hauptspeicher belegen. Beispielsweise hat **`sizeof(char)`** den Wert 1.
Geben Sie in einem C++ - Programm für jeden elementaren Datentyp die Größe des benötigten Speicherplatzes (zusammen mit dem Datentyp) auf dem Bildschirm aus.

b) Legen Sie eine Struktur "student" an mit folgenden Elementen:
- id
- name
- studiengang
- note
- bestanden

Überlegen Sie sich hierfür geeignete Datentypen für die einzelnen Elemente.
Geben Sie mit dem sizeof-Operator den Speicherplatz für die Struktur student aus.

c) Legen Sie ein Feld von 3 Variablen des Typs **`struct student`** an und initialisieren Sie die einzelnen Strukturelemente der Feldvariablen über eine Initialisierungsliste. Legen Sie anschließend zwei weitere Variablen vom Typ **`struct student`** an. In diesem Fall sollen die Werte der jeweiligen Stukturelemente jedoch einzeln über entsprechende Zuweisungen gespeichert werden.

Geben Sie anschließend die Werte der Elemente für alle 5 Strukturvariablen auf dem Bildschirm aus.

Die Lösung dieser Aufgabe soll sowohl die Implementierung als auch die Bildschirmausgabe des Programms umfassen.

Lösung:

```
#include <iostream>
#include <cstring> // für strcpy

using namespace std;


// Zu b) Anlegen von Struktur student
struct student
{
  int id;
  char name[80];
  char studiengang[10];
  float note;
  bool bestanden;
};


// Zu c) Deklaration der Ausgabefunktion für student
void ausgabe( const struct student *stud );
```

```
// Hauptfunktion
int main ( int argc, char* argv[] )
{
    // Zu a) Ausgabe des Speicherplatzes von elementaren Datentypen

    cout << "short: " << sizeof(short) << endl;
    cout << "int: " << sizeof(int) << endl;
    cout << "long: " << sizeof(long) << endl;
    cout << "long long: " << sizeof( long long ) << endl;
    cout << "unsigned int: " << sizeof(unsigned int) << endl;
    cout << "float: " << sizeof(float) << endl;
    cout << "double: " << sizeof(double) << endl;
    cout << "long double: " << sizeof( long double) << endl;
    cout << "char: " << sizeof(char) << endl;
    cout << "bool: " << sizeof(bool) << endl;

    // Zu b) Speicherplatz der Struktur student

    cout << "Struktur student: " << sizeof(student) << endl;
    cout << endl;

    // Zu c) Wertebelegung von Strukturvariablen

    // Array von 3 Variablen vom Typ struct student über Initialisierungsliste
    struct student stud[3] = { { 1, "Tick", "ET", 2.0, true }, { 2,
"Trick", "IT", 1.7, true }, { 3, "Track", "Inf", 2.3, true } };

    // 2 weitere Variablen vom Typ struct student über Zuweisung
    struct student stud1, stud2;
    stud1.id = 4;
    strcpy( stud1.name, "Panzerknacker" );
    strcpy( stud1.studiengang, "Maschbau" );
    stud1.note = 5.0;
    stud1.bestanden = false;
    stud2.id = 5;
    strcpy( stud2.name, "Ede" );
    strcpy (stud2.studiengang, "Faulenzen" );
    stud2.note = 5.3;
    stud2.bestanden = false;

    // Ausgabe der Variablen
    ausgabe( &stud[0] );
    ausgabe( &stud[1] );
    ausgabe( &stud[2] );
    ausgabe( &stud1 );
    ausgabe( &stud2 );

    return 0;

}
```

```
// Zu c) Definition der Ausgabefunktion
void ausgabe( const struct student *stud )
{
     cout << "Id: " << stud->id << endl;
     cout << "Name: " << stud->name << endl;
     cout << "Studiengang: " << stud->studiengang << endl;
     cout << "Note: " << stud->note << endl;
     cout << "Bestanden: " << ( (stud->bestanden) ? ( "ja" ) : (
"nein" ) ) << endl;
     cout << endl;
}
```

Programmausgabe:

```
short: 2
int: 4
long: 4
long long: 8
unsigned int: 4
float: 4
double: 8
long double: 12
char: 1
bool: 1
Struktur student: 104

Id: 1
Name: Tick
Studiengang: ET
Note: 2
Bestanden: ja

Id: 2
Name: Trick
Studiengang: IT
Note: 1.7
Bestanden: ja

Id: 3
Name: Track
Studiengang: Inf
Note: 2.3
Bestanden: ja

Id: 4
Name: Panzerknacker
Studiengang: Maschbau
Note: 5
Bestanden: nein

Id: 5
Name: Ede
Studiengang: Faulenzen
Note: 5.3
Bestanden: nein

Press any key to continue . . .
```

Aufgabe 3 (Referenzen):

a) Gegeben sei folgendes Programm:

```
int main ( int argc, char* argv[] )
{
  int x = 5;
  int& y = x;

  y = 9;
  int a[100];
  int &b = a[55];

  b = y;

  int *p = &y;
  int& z = *p;

  z = 33;
  …
}
```

Geben Sie die Werte von x, y, z und b am Ende des Programms an.

Lösung:

```
int main ( int argc, char* argv[] )
{
  int x = 5; // integer-Variable x wird angelegt und mit 5 initialisiert
  int& y = x; // y ist eine Referenz auf x (d.h. "anderer Name" für x)

  y = 9; // x = y = 9
  int a[100]; // Feld a mit 100 integer-Elementen wird angelegt
  int &b = a[55]; // b ist eine Referenz auf das Element a[55]

  b = y; // b (und damit a[55]) wird der Wert von y, also 9 zugewiesen; x = y = b = 9

  int *p = &y; // p ist ein Zeiger, der mit der Adresse von y (und damit von x) initialisiert
               // wird; x = y = b = *p = 9
  int& z = *p; // z ist eine Referenz auf den Wert von p; x = y = b = *p = z = 9
  z = 33; // z und damit der Wert von p wird auf 33 gesetzt; da p die Adresse von y enthält,
          // werden auch x und y auf 33 gesetzt; b wird jedoch nicht verändert und bleibt 9

  cout << "x = " << x << endl;
  cout << "y = " << y << endl;
  cout << "z = " << z << endl;
  cout << "b = " << b << endl;

  return 0;
}

// x = 33
// y = 33
// z = 33
// b = 9
```

b) Legen Sie die Struktur „Student“ aus Aufgabe 2b an sowie ein Feld von 3 Variablen des Typs **`struct student`**, das über eine Initialisierungsliste initialisiert wird (hierzu können Sie den Code aus Aufgabe 2b verwenden).

Legen Sie anschließend folgende Referenzen an:

i. Eine Referenz auf das Element „note“ des ersten Feldelements,
ii. Eine Referenz auf das mittlere Feldelement,
iii. Eine Referenz auf das Element „bestanden“ der Referenz aus ii.

Verwenden Sie diese Referenzen, um die entsprechenden Werte der Variablen auf dem Bildschirm auszugeben (Hinweis: Bei der Referenz auf die Struktur in ii. müssen die einzelnen Strukturelemente dereferenziert und einzeln ausgegeben werden).

Lösung:

```
#include <iostream>

using namespace std;

struct student
{
  int id;
  char name[80];
  char studiengang[10];
  float note;
  bool bestanden;
};

int main( int argc, char* argv[] )
{
    struct student stud[3] = { { 1, "Tick", "ET", 2.0, true }, { 2,
"Trick", "IT", 1.7, true }, { 3, "Track", "Inf", 2.3, true } };

    // Referenz auf Strukturelement des ersten Feldelements
    float &rStrukturElement = stud[0].note;
    cout << "Note: " << rStrukturElement << endl;
    cout << endl;

    // Referenz auf mittleres Feldelement
    struct student &rFeldElement = stud[1];
    cout << "Id: " << rFeldElement.id << endl;
    cout << "Name: " << rFeldElement.name << endl;
    cout << "Studiengang: " << rFeldElement.studiengang << endl;
    cout << "Note: " << rFeldElement.note << endl;
    cout << "Bestanden: " << rFeldElement.bestanden << endl;
    cout << endl;

    // Referenz auf Element einer referenzierten Struktur
    bool &rElementReferenzierterStruktur = rFeldElement.bestanden;
    cout << "Bestanden: " << rElementReferenzierterStruktur << endl;

    return 0;
}
```

Aufgabe 4 (Referenzen als Aufrufparameter):

Schreiben Sie ein C++-Programm, welches aus der Funktion **main** und der Funktion **addiere** besteht. In der Funktion **main** sollen drei lokale **int**-Variablen angelegt und mit einer Zufallszahl zwischen 1 und 100 initialisiert werden; die Werte der drei Variablen sollen auf dem Bildschirm ausgegeben werden. Anschließend wird die Funktion **addiere** aufgerufen. **addiere** hat drei Aufrufparameter (Parameter 1: der Wert der ersten Zufallszahl-Variablen, Parameter 2: die *Adresse* der zweiten Zufallszahl-Variablen, Parameter 3: eine Referenz auf die dritte Zufallszahl-Variable) und gibt die Summe der Werte zurück. Außerdem werden lokal in der Funktion **addiere** (*nach* Berechnung der Summe) alle Werte verdoppelt und die verdoppelten Werte auf dem Bildschirm ausgegeben. Nach Aufruf der Funktion **addiere** wird dann in der **main** –Funktion der Wert der Summe ausgegeben sowie die Werte der drei Variablen.

Diskutieren Sie kurz das Ergebnis.

Lösung:

```
#include <iostream>
#include <ctime> // für time
#include <cstdlib> // für srand, RAND_MAX
using namespace std;

int addiere( int x, int* y, int& z );

int main (int argc, char* argv[] )
{
    int MAX = 100;

      srand( time(NULL) );

      int x = rand() % MAX + 1;
      int y = rand() % MAX + 1;
      int z = rand() % MAX + 1;

      cout << "x = " << x << ", y = " << y << ", z = " << z << endl
<< endl;

      int sum;

      sum = addiere( x, &y, z );

      cout << "Die Summe lautet: " << sum << endl << endl;

      cout << "nach addiere:" << endl;
      cout << "x = " << x << ", y = " << y << ", z = " << z << endl
<< endl;

      system("PAUSE");

      return 0;
}
```

```
int addiere( int x, int* y, int& z )
{
      int sum = x + *y + z;

      x *= 2;
      *y *= 2;
      z *= 2;

      cout << "in addiere:" << endl;
      cout << "x = " << x << ", y = " << *y << ", z = " << z << endl
<< endl;

      return sum;
}
```

Programmausgabe:

```
x = 96, y = 72, z = 67

in addiere:
x = 192, y = 144, z = 134

Die Summe lautet: 235

nach addiere:
x = 96, y = 144, z = 134

Press any key to continue . . .
```

Diskussion:

Der erste Aufrufparameter von **addiere** realisiert call-by-value, d.h. eine Veränderung der Variablen in **addiere** ist in der Hauptfunktion nicht sichtbar. Der zweite Aufrufparameter von **addiere** realisiert call-by-reference über einen Zeiger, d.h. Veränderungen in **addiere** sind in der Hauptfunktion sichtbar. Allerdings wird in **addiere** die Dereferenzierungssyntax benötigt. Der dritte Aufrufparameter schließlich realisiert ebenfalls call-by-reference über eine Referenz. Wiederum sind Veränderungen der Variablen in **addiere** in der Hauptfunktion sichtbar. Hier entfällt jedoch die Dereferenzierungssyntax in **addiere**. Dadurch ist besondere Vorsicht beim Programmieren und Lesen von Code erforderlich, denn call-by-reference teilt sich hier nur durch das Symbol „&“ in der Deklaration und Definition von **addiere** im dritten Aufrufparameter mit. Falls keine Veränderungen der Variablen in der aufgerufenen Funktion stattfinden, sollte daher mit **const** gearbeitet werden.