Nachname: ..... Vorname: ..... MatrNr.: .....

Klausur Programmieren 1

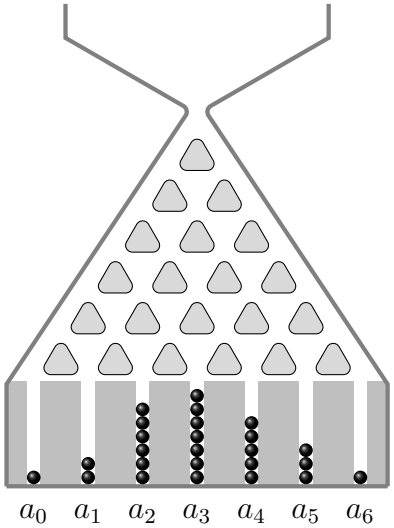
HAW-Hamburg, Fakultät Technik und Informatik, Department Informations- und Elektrotechnik
Prof. Dr. Robert Heß, 11.7.2017, Bearbeitungsdauer: 180 Min.
Erlaubte Hilfsmittel: Vorlesungsunterlagen, Lösungen aus dem Praktikum und C/C++ Einführungsbücher.

Ergebnis: ..... von 100 Punkten Note: ..... Punkte.

---

1 Einleitung

1.1 Das Thema



Francis Galton entwarf im vorletzten Jahrhundert das sog. *Galton Brett*, mit dem die Binomialverteilung von Zufallsprozessen veranschaulicht werden kann: Eine Kugel trifft auf der ersten/obersten Ebene auf ein Hindernis, und bewegt sich zufällig rechts oder links weiter. Danach trifft die Kugel auf ein weiteres Hindernis und entscheidet sich wieder per Zufall, ob es den rechten oder linken Pfad wählt. Dieser Vorgang wiederholt sich mehrfach, bis die Kugel in einen Behälter fällt.

Wird dieser Vorgang mit einer großen Anzahl an Kugeln wiederholt, so ergeben die Füllhöhen der Auffangbehälter eine Binomialverteilung. Die unterliegende Mathematik lernen Sie im 3. Semester, aber hier wollen wir uns dem Thema mit einem nummerischen Experiment nähern.

Es soll ein Programm erstellt werden, bei dem der Benutzer die Anzahl der Sammelbehälter (1-20) und die Anzahl der Kugeln (10-10 000) angeben kann. Das Programm führt das Experiment entsprechend oft aus und ermittelt den Mittelwert und die Standardabweichung der ermittelten zufälligen Verteilung.

1.2 Zufallszahlen

C stellt uns in der Header-Datei *stdlib.h* einen für unsere Zwecke brauchbaren Generator für Zufallszahlen zur Verfügung: Der Befehl *rand()* liefert eine ganze Zahl im Bereich null bis *RAND_MAX*. (*RAND_MAX* ist mit *#define* in der Header-Datei *stdlib.h* als Makro definiert.)

Für das Galton-Brett muss per Zufall ermittelt werden, in welchen Behälter die Kugel fällt. Für jede Ebene wird dabei ermittelt, ob die Kugel rechts oder links vom Hindernis nach unten fällt. Die Entscheidung kann mit der Funktion *rand()* erfolgen: Ist der Zahlenwert größer als *RAND_MAX*/2 fällt die Kugel nach rechts, ansonsten fällt sie nach links.

Es bietet sich an, die Behälter mit Nummern beginnend mit null zu versehen. Der Index für den Behälter ergibt sich dann aus der Anzahl, wie oft die Kugel nach rechts gefallen ist.

Werden keine Gegenmaßnahmen getroffen, so liefert die Funktion *rand()* bei jedem Programmstart die selben Zufallszahlen. Um das zu verhindern, steht die Funktion *srand()* zur Verfügung, die eine ganze Zahl zum Initialisieren des Zufallszahlengenerators erwartet. Das zu erstellende Programm soll zu Beginn eine ganze Zahl (0-999), den *Seed*-Wert vom Benutzer abfragen, mit der dann der Generator mittels der Funktion *srand()* initialisiert wird.

1.3 Statistik

Zur statistischen Auswertung bekommen die Kugeln der Behälter a_k den Wert k. Es sollen Erwartungswert μ, Varianz σ^2 und Standardabweichung σ bestimmt werden. Mit N als die Anzahl der Kugeln, n als die Anzahl der Behälter und a_k, $k = 0 \ldots n-1$ als die Anzahl der Kugeln in den Behältern folgt:

$$\mu = \frac{1}{N} \sum_{k=0}^{n-1} a_k \, k \qquad \sigma^2 = \frac{1}{N-1} \sum_{k=0}^{n-1} (k-\mu)^2 \, a_k \qquad \sigma = \sqrt{\sigma^2}$$

2 Programmieraufgaben

Aufgabe 1 (10 Punkte)

Erstellen Sie eine Funktion mit Namen *getIntMinMax* zur sicheren Abfrage einer ganzen Zahl. Die Funktion erwartet einen Fragetext, einen Minimalwert und einen Maximalwert. Fehlerhafte Eingaben (Buchstaben, zu kleine oder zu große Werte) werden mit Fehlermeldung und erneuter Abfrage abgefangen.

Aufgabe 2 (20 Punkte)

Erstellen Sie eine Funktion mit Namen *evalDistribution* mit Rückgabetyp *void*, die folgende Parameter erwartet:

- Vektor für die Anzahl der Kugeln in den Behältern a_0 bis a_{n-1}
- ganze Zahl für die Anzahl der Behälter n
- ganze Zahl für die Gesamtzahl der Kugeln N

Für jede der N Kugeln wird $n-1$ mal per Zufall eine Entscheidung getroffen, ob die Kugel nach links oder nach rechts fällt. Der Index des Behälters ergibt sich aus der Anzahl der Entscheidungen nach rechts. (Am Ende muss die Summe der Zahlen in den Behältern a_k der Gesamtzahl der Kugeln N entsprechen: $\sum_{k=0}^{n-1} a_k = N$)

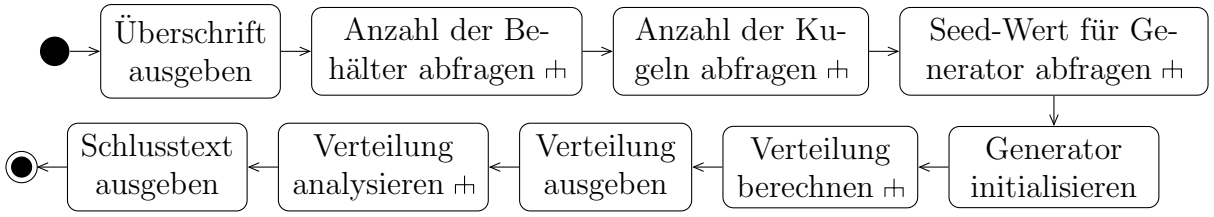
Aufgabe 3 (15 Punkte)

Erstellen Sie eine Funktion mit Namen *analyseDistribution*, die keinen Wert zurückgibt und die selben Parameter wie *evalDistribution* erwartet.

Die Funktion ermittelt Erwartungswert μ, Varianz σ^2 und Standardabweichung σ gemäß den in Abschnitt 1.3 angegebenen Formeln und gibt die Werte auf dem Bildschirm aus. Zum Ziehen der Wurzel verwenden Sie die Funktion *sqrt()* aus der Header-Datei *math.h*.

Aufgabe 4 (25 Punkte)

Fügen Sie die Programmstücke zu einem lauffähigen Programm zusammen:



Nachname: ..... Vorname: ..... MatrNr.: .....

Achten Sie auf einen guten Programmierstil (Vermeidung globaler Variablen, sinnvolle Variablennamen, Quellcode einrücken und kommentieren, keine absoluten Sprünge mit **goto**, keine Warnungen vom Compiler etc.).

3 Verständnisfragen

Aufgabe 5 (6 Punkte)

Eine Benutzerabfrage soll solange wiederholt werden, bis die Eingabe die geforderten Voraussetzungen erfüllt. Welche Schleife ist zu bevorzugen? Begründen Sie Ihre Antwort.

☐ **for** ... ☐ **while** ... ☐ **do** ...

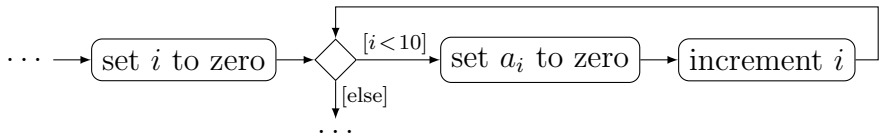
Aufgabe 6 (6 Punkte)

Erläutern Sie folgenden Sprachkonstrukt und gehen Sie dabei auf die Ausdrücke 1 bis 3 ein.

<Ausdruck1>?<Ausdruck2>:<Ausdruck3>

Aufgabe 7 (6 Punkte)

Das folgende Aktivitätsdiagramm initialisiert einen Vektor mit null. Was ist falsch in dem Diagramm?



Aufgabe 8 (6 Punkte)

Stellen Sie folgende Ausdrücke dual/binär dar:

a) $4 << 4 \Rightarrow$ *b*) $15 >> 2 \Rightarrow$ *c*) $2^8 - 1 \Rightarrow$

Aufgabe 9 (6 Punkte)

Welchen Datentyp haben folgende Ausdrücke?

1/2	
cos(0)/2	
1/2.0f	

0xAffe	
1.	
'1'	