Programmieren für Physiker

Interfakultatives Institut für Anwendungen der Informatik
Institut für Theoretische Teilchenphysik

Prof. Dr. M. Steinhauser, Dr. A. Mildenberger
http://comp.physik.kit.edu

SS 2023 – Blatt 08
Bearbeitung bis 21. Juni 2023

Aufgabe 20: Rechteck-Klasse **Pflichtaufgabe**

Es ist eine Klasse zur Beschreibung von Rechtecken zu schreiben. Rechtecke sind dabei durch zwei Gleitkommazahlen, also Breite und Höhe charakterisiert. Diese Variablen sollen von außen nicht zugänglich sein.
Programmieren Sie bitte die Klasse mit den folgenden drei Methoden:

(a) `setze`. Die Methode habe zwei `double`-Argumente, die Breite und Höhe des Rechtecks zugewiesen werden.
(b) `rotiere`. Die Methode habe kein Argument. Sie vertauscht in der Klasseninstanz Breite und Höhe, rotiert also um 90°.
(c) `bedeckt`. Diese Methode hat ein weiteres Rechteckargument und gibt per Logikwert zurück, ob das Rechteck der eigenen Klasseninstanz das Rechteck im Argument komplett bedecken kann. Dabei genügt es, die Rechtecke in der gegebenen Orientierung zu betrachen und lediglich die Höhen und Breiten miteinander zu vergleichen.

Auf der Kurswebseite findet sich ein Hauptprogramm, das die Klasse einsetzt. Ergänzen Sie bitte den Code, dass das gegebene Hauptprogramm funktioniert.

Aufgabe 21: Quicksort **Pflichtaufgabe**

Quicksort (C.A.R. Hoare, 1961/62) ist ein schneller, rekursiver Sortieralgorithmus. Hier eine Skizze der Funktionsweise: Es wird aus der zu sortierenden Liste ein „Pivotelement“ m gewählt (im einfachsten Fall kann das erste Element der Liste benutzt werden) und die Liste wird in einem Durchgang in zwei Teillisten geordnet. In der linken Teilliste sind alle Elemente $\leq m$, alle Elemente der rechten Teilliste sind $\geq m$. Dieser Schritt heißt auch Partitionierung. Nun können die beiden noch unsortierten Teillisten jeweils für sich mit dem gleichen Verfahren bearbeitet werden, dies geschieht rekursiv. Teillisten mit weniger als zwei Elementen brauchen natürlich nicht mehr sortiert zu werden.
Schreiben Sie eine Funktion `QuickSort (double A[], int s, int t)`, die im Feld `A` zwischen den Indizes `s` und `t` sortiert. Bei Bedarf soll diese Funktion eine Partitionierung durchführen und sich selbst aufrufen. Entwerfen Sie eine eigene Partitionierung oder verwenden Sie folgendes Schema:

```
Partitionierung (A, s, t)
  pivot := A[s]
  l := s
  Schleife i von s+1 bis t
    Falls A[i] < pivot
    dann:  l := l+1
           Tausche (A[i],A[l])
  Tausche (A[s],A[l])
```

Wie funktioniert diese Partitionierung? Wo steht nach Ausführung dieser Routine das Pivotelement, welche beiden Teillisten sind nun also zu sortieren?

Um Ihre Sortierfunktion zu testen, generieren Sie ein Feld mit 1000 zufälligen `double`-Werten im Bereich $[0, 1]$.
Die Zufallszahlen können Sie folgendermaßen erzeugen: Mit den Anweisungen `#include <cstdlib>` und `#include <ctime>` im Kopf des Programms ergibt `double(rand())/RAND_MAX` bei jedem Aufruf eine gleichverteilte Zufallszahl im Intervall $[0, 1[$. Um den Generator zu Beginn auf einen zufälligen Startwert zu setzen, benötigen Sie noch ein einmaliges `srand((unsigned int)time(0))` am Anfang des Programmcodes.

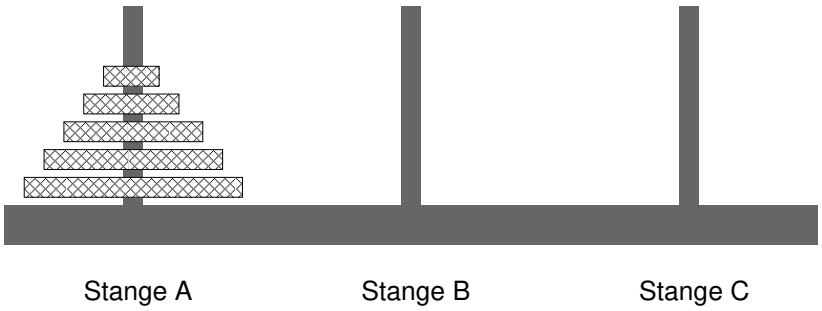
Verwenden Sie zum Tausch von Feldelementen und für die Partitionierung jeweils Funktionen.

Überlegen Sie sich bitte, wie viele und welche Modifikationen in Ihrem Programm nötig sind, um die Zahlenliste absteigend statt aufsteigend zu sortieren.

Zusatzfragen (freiwillig): In welchen Ausgangssituationen ist die Laufzeit von Quicksort in der obigen Version am ungünstigsten, d.h. sind am meisten Vergleiche durchzuführen? Messen Sie die Laufzeit des Programms mit geeignet langen Zahlenfeldern für verschiedene Ausgangssituationen.

Aufgabe 22: Türme von Hanoi **freiwillig**

Das Spiel *Türme von Hanoi* besteht aus n verschieden großen Scheiben und drei Stangen. Zu Beginn liegen alle Scheiben, der Größe nach sortiert, auf der ersten Stange. Ziel des Spieles ist es, alle Scheiben von der ersten auf eine andere Stange zu versetzen. Dabei sind jedoch folgende Regeln zu beachten: Es darf immer nur eine Scheibe zu einem Zeitpunkt bewegt werden und es darf nie eine größere auf eine kleinere Scheibe gelegt werden. Eine Skizze der Ausgangsposition mit 5 Scheiben:



Um ein Gefühl für die Problemstellung zu erhalten, ist es ratsam, sich zunächst die Lösung für $n = 1$, 2, 3 und evtl. $n = 4$ zu überlegen.
Ist Ihnen aufgefallen, dass Sie das Vorgehen für n Scheiben auf die Lösung für $n - 1$ Scheiben zurückführen können? Nehmen Sie an, Sie hätten ein Verfahren, um die obersten $n - 1$ Scheiben zu versetzen. Welche Schritte sind dann nötig, um den Turm der Größe n zu versetzen? Dieser Lösungsweg kann mittels Rekursion in einem Programm umgesetzt werden.
Entwickeln Sie ein Programm, das für ein einzugebendes n alle einzelnen Scheibenbewegungen ausgibt, die nötig sind, um das Problem zu lösen.
