Programmierstarthilfe SS 2008
Fakultät für Ingenieurwissenschaften und Informatik

11. Blatt
Für den 7.7. und 8.7.2008

Organisatorisches

Um auf die Mailingliste aufgenommen zu werden schicke einfach eine Mail an `guido.de-melo@uni-ulm.de`.

Die Webseiten zur Veranstaltung sind unter `http://www.uni-ulm.de/in/mi/lehre/ss2008/programmierstarthilfe.html` zu finden.

ACM Programmierwettbewerb

Am 11. Juli findet wieder der ACM Programmierwettbewerb an der Universität Ulm statt. Der Wettbewerb dient als lokaler Ausscheidungswettbewerb für die Qualifikation zum South West European Regional Contest (SWERC). Wenn ihr Spass am Programmieren habt und vielleicht die langjährige Erfolgsgeschichte Ulmer Programmiermannschaften weiterführen wollt, könnt ihr euch unter `http://www.informatik.uni-ulm.de/acm/Locals/2008/` für den Wettbewerb anmelden. Die Teilnahme ist kostenlos aber auf keine Fall umsonst. :)

Auch Anfänger sind willkommen, um ein wenig Wettbewerbsluft zu schnuppern.

Wenn ihr euch den Typ der Aufgaben anschauen und vielleicht ein wenig üben wollt, könnt ihr euch auf der Seite `http://www.spoj.pl` registrieren. Hier gibt es Aufgaben verschiedener Schwierigkeitsgrade, die in einer von über 30 möglichen Programmiersprachen gelöst werden können.

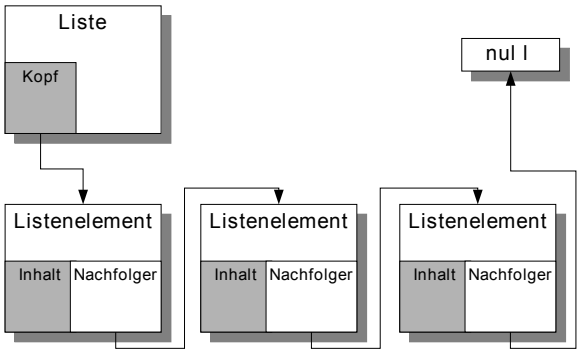
Weitere Informationen über vergangene Wettbewerbe und Aktivitäten der Programmiermannschaft findet ihr unter `http://www.informatik.uni-ulm.de/acm/`.

1. Listen

In der Informatik existieren neben den *primitiven Datentypen*, die wir in den letzten Wochen kennengelernt haben, noch sogenannte *abstrakte Datentypen*. Diese abstrakten Datentypen sind fortschrittliche Datentypen, die für viele Algortihmen und Problemlösungen benötigt werden. Im Grunde sind abstrakte Datentypen zunächst Beschreibungen, wie diese Datenstrukturen aufgebaut sind, zu was sie dienen und wie sie funktionieren. Allerdings existieren in vielen Programmiersprachen wie auch in Java bereits vorimplementierte fortschrittliche Datentypen.
Doch in in einem Informatikstudium sollte man auch selbst einmal einfache abstrakte Datentypen implementiert haben. Ein einfacher abstrakter Datentyp ist die sogenannte Liste, die

ähnlich einem Array mehrere Elemente eines bestimmten Typs aufnehmen kann. Anders als bei einem Array kann eine Liste aber eine variable Anzahl von Elementen aufnehmen und somit kann auch die Länge der Liste dynamisch geändert werden.
Weitere wichtige Datentypen sind unter anderem Stapel (Stacks), Schlangen (queues), Bäume (trees), Halden (heaps) oder assoziative Arrays (hash maps). Das Thema der abstrakte Datentypen wird im Wintersemester in der Vorlesung *Algorithmen und Datenstrukturen* vertieft.

Listen bestehen aus einzelnen Listenelementen. Eine Referenz auf das erste Element einer Liste wird in der Liste als Kopf gespeichert, damit der Anfang der Liste definiert ist. Die einzelnen Listenelemente referenzieren wiederum auf das nachfolgende Listenelement. Eine wichtige Ausnahme stellt das letzte Element dar, dass keinen weiteren Nachfolger besitzt und deswegen auf `null` zeigt.
Wie bei vielen abstrakten Datentypen gibt es auch bei den Listen diverse Variationen. So gibt es auch Listen mit Listenelementen, die ihren Nachfolger und ihren Vorgänger verlinken, oder auch Listen, bei denen das letzte Element wieder auf den Listenkopf zeigt, und somit einen Ring erzeugt. Im Folgenden beschränken wir uns aber auf einfach verkettete Listen, die hier nochmal schematisch skizziert sind:



2. Aufgaben

Im Folgenden möchten wir Schritt für Schritt eine eigene Listenstruktur programmieren und damit arbeiten. Die folgenden Aufgaben bauen größtenteils aufeinander auf und sollten deswegen auch in richtiger Reihenfolge bearbeitet werden.

2.1. Klassen für Listenelemente und Listen

Zunächst benötigen wir zwei Klassen. Eine Klasse soll später einzelne Listenelemente darstellen, eine zweite Klasse dann die Liste selbst. Unsere Listenelmente sollen jeweils einen Integer-Wert speichern können, also stellt die fertige Liste später eine Liste von Integer-Werten dar.

a) Erstelle die Klasse `ListenElement`. Es soll zwei Objekt-Attribute besitzen: Einen `int`-Wert inhalt, sowie einen Attribut vom Typ `ListenElement` und dem Namen nachfolger. Das Attribut Inhalt speichert später die Zahl, die das Listenelement speichern soll, Nachfolger ist eine Referenz auf das nächste Element in der Liste.

b) Teste nun deine Klasse in einer Main-Methode und erstelle hierfür zwei Listenelemente und setze das zweite Element als Nachfolger des ersten.

c) Desweiteren soll nun eine Klasse `Liste` geschrieben werden, die ein Atrribut kopf vom Typ `ListenElement` besitzen soll. Diese Klasse soll uns später helfen, mit einer kompletten Liste zu arbeiten, ohne dabei alle einzelnen Listenelemente einzeln abzuspeichern.

d) Erzeuge nun in deiner Testmethode ein neues Objekt vom Typ Liste und weise dem Attribut kopf dein ersten Listenelement zu.

2.2. Methoden für Listenelemente und Listen

Bisher haben wir nur mit Attributen gearbeitet, da die Listen und Listenelemente aber Objekte sind, können wir nicht nur Daten darin speichern, sondern auch Methoden und Konstruktoren für sie anlegen. Dies wollen wir nun machen.

a) Die Klasse `ListenElement` soll nun einen Konstuktor bekommen, in dem man einen Integer-Wert übergibt. Dabei soll dann das neu angelegte Objekt vom Typ ListenElement direkt diesen Wert als inhalt-Attribut gesetzt bekommen.

b) Erweitere ebenso die Klasse `Liste`, die im Konstruktor ein Listen-Element übergeben bekommen soll, welches dann den direkt den Kopf der Liste setzt.

c) Die Klasse `ListenElement` benötigt außerdem eine Objektmethode `setNachfolger(ListenElement nachfolger)`, womit der Nachfolger des Listenelements gesetzt werden kann.

d) Schreibe außerdem eine Methode `getEnde()` für die Klasse `Liste`, die das letzte in der Liste enthaltenene Element zurückgeben soll.

e) Nun folgt eine Methode `fuegeHinzu(ListenElement nachfolger)` für die Klasse `Liste`, die ein übergebenes Listenelement an das Ende der Liste anhängen soll.

2.3. Typische Operationen mit Listen

Nun wollen wir wichtige Funktionen von Listen hinzufügen.

a) Schreibe eine Methode `ausgabe()` für die Klasse `Liste`, die die einzelnen Integer-Werte der in der Liste enthaltenen Elemente hintereinander ausgibt. Überlege dir zunächst, wie du sinnvoll die einzelnen Elemente der Liste durchgehen kannst.

b) Schreibe eine Methode `anzahl()` für die Klasse `Liste`, die zurückgeben soll, wie viele Elemente sich in der Liste befinden.

c) Schreibe eine Methode `istEnthalten(int wert)` für die Klasse `Liste`, die true zurückgeben soll, wenn sich in der Liste ein Listenelemnt mit dem übergenen Wert befindet, und ansonsten false zurückgibt.

d) Kopiere und erweitere die vorherige Methode `istEnthalten(int wert)` zu `suche(int wert)`. Die Suchmethode soll das Listenelement zurückgeben, falls es ein solches Element mit dem übergebenen Wert gibt, und ansonsten null zurückgeben.

e) Eine Methode `loesche(int wert)` in der Listenklasse soll Elemente aus der Liste löschen, die diesen Wert besitzen. Hierbei soll aber die restliche Liste unberührt bleiben. Achte also darauf, dass du beim Löschen von Werten nicht die Liste zerstörst.

3. Bonusaufgaben

3.1. Fortgeschrittene Aufgaben zu Listenelemente und Listen

a) Schreibe eine weitere Variante der Methode `anzahl()`, die die Anzahl der Elemente in der Liste zählt. Hierbei sollst du dieses Mal rekursiv, und nicht iterativ zählen. Überlege dir zunächst auf einem Blatt Papier die typischen Merkmale von rekursiven Algorithmen (Basisfall, Rekursionfall etc.) für dein spezielles Problem.

b) Eine Methode `bildeDurchschnitt()` in der Klasse `Liste` soll die Liste durchlaufen, den Durchschnittswert aller enthaltenen Zahlenwerte bilden, und als float-Wert zurückgeben. Hier kannst du wahlweise iterativ oder rekursiv vorgehen.

c) Schreibe eine Methode in der Klasse `Liste`, um die Liste komplett umzudrehen. Hierfür kannst du dir hilfsweise mit einer zweiten Liste arbeiten.

d) Schreibe eine statische Methode, die zwei Listen entgegennimmt und überprüft, ob beide Listen den gleichen Inhalt besitzen.

e) Insertion Sort ist ein Sortieralgorithmus, der insbesondere mit Listen einfach zu implementieren ist. Informiere dich im Internet, wie Insertion Sort funktioniert, und implementiere eine Methode `sortiere()` für die Klasse `Liste`.