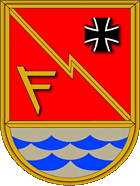
Führungsunterstützungsschule

der Bundeswehr



**Sprachausbildung Java**



**Übungen**

**"Dynamische Datenstrukturen"**

Listen

Aufgabe 1

Mit dieser Aufgabe wollen wir eine Grundlage für den Umgang und das Arbeiten mit Listen schaffen. Unsere Listenelemente sind wie folgt definiert:

class ListElem{

private int key;

private int info;

private ListElem next;

}

Im Teil key sollen die Listenelemente von vorn nach hinten durchnummeriert sein, beginnend bei 0. In info steht der Wert des jeweiligen Elements und in next das Nachfolger-Element, wenn kein Nachfolger existiert ist dieser mit dem Wert null belegt.

Für derartige Listen wollen wir einen Satz nützlicher Funktionen bereitstellen. Achten Sie darauf, dass in den Funktionen die originalen Listen nicht zerstört werden. (Arbeiten Sie wo möglich mit Kopien der Listen, siehe Funktion aus b))

1. Schreiben Sie eine Funktion readIn die Werte einließt und damit eine Liste nach dem FIFO-Prinzip erstellt.
2. Schreiben Sie eine Funktion copy, die eine Liste kopiert. Damit ist aber nicht gemeint, dass der Startzeiger kopiert wird, sondern eine neue Liste mit den neuen Elementen aber gleichen Werten aufgebaut wird.
3. Schreiben Sie eine Funktion show, die die Liste vernünftig ausgibt.
4. Schreiben Sie eine Funktion conc, mit der zwei Listen miteinander verknüpft werden. Dabei soll die zweite Liste einfach an das letzte Element der ersten Liste angehängt werden.
5. Schreiben Sie eine Funktion head, die nur das erste Listenelement zurückliefert und den Rest abschneidet. Das heißt der next-Zeiger dieses Elements zeigt auf null.
6. Schreiben Sie eine Funktion tail, die das erste Listenelement abschneidet und den Rest der Liste zurückliefert.
7. Schreiben Sie eine Funktion length, die die Länge der Liste liefert.
8. Schreiben Sie eine Funktion add(int info, int key), die ein Elementwert und einen Schlüsselwert key bekommt. Es soll das info Attribut des Elements an der Stelle key mit dem übergebenenen info-Wert überschreiben. Falls kein passendes Element existiert, muss nichts gemacht werden.
9. Schreiben Sie eine Funktion get(int key), die einen Schlüsselwert bekommt und die info an dieser Stelle zurückliefert. Sollte der key-Wert nicht existieren wird -1 zurückgegeben.
10. Schreiben Sie eine Funktion drop, die einen Schlüsselwert bekommt und das Element an dieser Stelle entfernt.
11. Schreiben Sie eine Funktion reverse, die eine Liste umkehrt, d.h. von hinten nach vorn verkettet.

Aufgabe 2 - Flavius-Methode

Der jüdische Geschichtsschreiber Flavius Josephus berichtet, dass er zusammen mit 40 anderen Juden in einen Keller geflüchtet sei, als die Römer eine jüdische Stadt eroberten. Um den Feinden nicht in die Hände zu fallen, wollten sie sich gegenseitig umbringen. Josephus war dagegen, konnte die anderen aber nicht umstimmen. Um dem Tod zu entgehen, schlug er vor, man solle sich im Kreis aufstellen und reihum abzählen. Jeder Dritte sollte auf der Stelle umgebracht werden. Sein Vorschlag wurde akzeptiert, und Josephus stellte sich natürlich so in den Kreis, dass er als letztes übrig und so am Leben blieb.

Schreiben Sie ein vollständiges Java-Programm, dass eine Folge von Namen für die beteiligten Personen einliest. Die Liste soll die Namen ohne überflüssige Leerzeichen in der Reihenfolge des Einlesens enthalten.

Beenden Sie das Einlesen, wenn das Zeichen **'\_'** einzeln als Name angegeben wurde. Schließen Sie sodann den Kreis und beginnen Sie mit dem Abzählen bei der Person, die als erstes eingelesen wurde.

Entfernen Sie nacheinander die jeweils dritte Person aus dem Kreis und geben Sie deren Namen auf dem Bildschirm aus.

Aufgabe 3

Stellen Sie Polynome (über einer Variable x) durch lineare Listen mit Elementen folgenden Typs dar:

class PolyGlied{

private int exponent;

private float koeffizient;

private PolyGlied naechster;

}

In der Polynom-Liste sollen die Glieder stets nach fallendem Exponenten angeordnet sein. Keine zwei Glieder dürfen den gleichen Exponenten haben. Glieder mit Koeffizient 0 werden nicht angegeben.

Bsp:

*f* (x) = 14.3 x4 + 3.7 x2 - 10.9 x + 4.0

wird wie folgt gespeichert:



* Schreiben Sie eine Methode drucke die ein Polynom "wie üblich" textuell darstellt und eine Methode auswerte, die das Polynom für einen ebenfalls gegebenen x-Wert ausrechnet.

(Beispiel Liste eines Polynoms{2,3,{2,1,{0,4,NULL}}}und Ausgabe: 3\*x2+2\*x1+4)

* Schreiben Sie eine Methode add, die zu zwei Polynomen das Summenpolynom berechnet.

Bsp:

2\*x³+3\*x²+3\*x+1 + 4\*x²+2\*x+2 = 2\*x³+(3+4)\*x²+(3+2)\*x+(1+2) = 2\*x³+(7)\*x²+(5)\*x+(3)

* **Zusatz**: Schreiben Sie eine Funktion mult, zur Berechnung des Polynomprodukts.

Bsp: 2\*x²+1 \* 3\*x+2 = 2\*3\*x²\*x+2\*2\*x²+1\*3\*x+1\*2 = 6\*x³+4\*x²+3\*x+2

Aufgabe 4

1. class ListElem{
2. private int key;
3. private int info;
4. private ListElem next; //Nachfolger
5. private ListElem pre; //Vorgänger
6. };

In dieser Aufgabe wollen wir die Liste mit den uns schon bekannten Methoden aus Aufgabe 1 etwas erweitern. Wir betrachten eine andere Listenstruktur, nämlich die der doppelt verketteten Listen. Der Unterschied liegt darin, dass neben dem Zeiger zum nächsten Element auch ein Zeiger zum vorhergehenden Element existiert:

Im Teil key sollen die Listenelemente von vorn nach hinten durchnummeriert sein, beginnend bei 0. In info steht der Wert des jeweiligen Elements und in next das Nachfolger-Element, wenn kein Nachfolger existiert ist dieser mit dem Wert null belegt. In pre soll der Zeiger zum Vorgänger enthalten sein. Dieser ist null, wenn es keinen Vorgänger gibt, was aber nur für das erste Element der Liste der Fall ist.

Für derartige Listen wollen wir einen Satz nützlicher Methoden bereitstellen. Achten Sie darauf, dass in den Methoden die originalen Listen nicht zerstört werden. (Arbeiten Sie wo möglich mit Kopien der Listen, siehe Methode aus b))

1. Schreiben Sie eine Methode readIn die Werte einließt und damit eine Liste nach dem FIFO-Prinzip erstellt.
2. Schreiben Sie eine Methode copy, die eine Liste kopiert. Damit ist aber nicht gemeint, dass der Startzeiger kopiert wird, sondern eine neue Liste mit den neuen Elementen aber gleichen Werten aufgebaut wird.
3. Schreiben Sie eine Methode show, die die Liste vernünftig ausgibt.
4. Schreiben Sie eine Methode conc, mit der zwei Listen miteinander verknüpft werden. Dabei soll die zweite Liste einfach an das letzte Element der ersten Liste angehängt werden.
5. Schreiben Sie eine Methode head, die nur das erste Listenelement zurückliefert und den Rest abschneidet. Das heißt der next-Zeiger dieses Elements zeigt auf null.
6. Schreiben Sie eine Methode tail, die das erste Listenelement abschneidet und den Rest der Liste zurückliefert.
7. Schreiben Sie eine Methode length, die die Länge der Liste liefert.
8. Schreiben Sie eine Methode replace(int wert, int key), die einen Elementwert und einen Schlüsselwert key bekommt. Es soll das neue Element an die Stelle key eingefügt werden.
9. Schreiben Sie eine Methode get(int key), die einen Schlüsselwert bekommt und das Element an dieser Stelle zurückliefert.
10. Schreiben Sie eine Methode drop, die einen Schlüsselwert bekommt und das Element an dieser Stelle entfernt.
11. Schreiben Sie eine Methode reverse, die eine Liste umkehrt, d.h. von hinten nach vorn verkettet.

Binärbäume

Aufgabe 5

Zunächst erstmal eine kleine Übung zum besseren Verständnis für Bäume.

Fügen Sie in einen (anfangs) leeren binären Suchbaum folgende Datensätze ein. Diese bestehen jeweils aus einem Schlüssel und einem Namen.

(50, Hans-Ulrich), (24, Arthur), (10, Armin),

(60, Carsten), (53, Christoph), (93, Felix),

(97, Tobias), (65, Maria), (62, Valeriy),

(52, Dharlakshmi), (63, Christopher), (51, Steffi)

Zeichnen Sie sich zur Hilfe zunächst den Baum auf, um die folgenden Fragen zu beantworten:

* Welche Suchpfade entstehen bei der Suche nach den Schlüsseln *50, 51, 65* und *66*?
* Löschen Sie nacheinander die Schlüssel 10, 53 und 60. Wie sieht der resultierende Baum aus?
* Geben Sie die Höhe des resultierenden Baumes an!

Aufgabe 6 – Blätter zählen

Als Blätter in einem Binärbaum werden die Knoten bezeichnet, die keine Nachfolger haben.

Folgender Binärbaum hat z.B. 4 Blätter:



Schreiben Sie eine Methode, die zu einem beliebigen Binärbaum die Anzahl der Blätter liefert, sowie eine Methode, die die Knoten zählt.

public int anzahl\_Blaetter(TBaumElem k);

public int anzahl\_Knoten(TBaumElem k);

Aufgabe 7 – Höhe eines Baumes

Die Höhe eines Binärbaumes ist definiert als die Anzahl der Knoten des längsten Astes.

Die Höhe kennzeichnet damit auch die maximale Verzweigungstiefe des Binärbaumes.

Schreiben Sie, eine Methode, die zu einem beliebigen Binärbaum dessen Höhe liefert.

public int hoehe(TBaumElem k);

Aufgabe 8 – Interessante Methoden

Zum jetzigen Zeitpunkt sollten Sie Binärbäume ausreichend kennen- und schätzen gelernt haben, so dass wir uns mit etwas interessanteren Methoden beschäftigen können. Implementieren Sie dazu folgende Methoden:

* TBaumElem conc(TBaumElem baum, TBaumElem anderer) konkateniert (fügt zusammen) einen anderen Baum an den aktuellen.
* void printPretty(TBaumElem baum) die einen Baum schön ausgibt, so dass direkt dessen Baumstruktur erkennbar ist.
* TBaumElem tausche(TBaumElem baum) tauscht den linken und den rechten Teilbaum.