Friedrich-Schiller-Universität Jena Fakultät für Mathematik und Informatik Institut für Informatik Lehrstuhl für Programmiersprachen und Compiler	Höhere Programmierung SS 2001	Praktikumsblatt 6 Ausgabe: 11.06.2001 Termin: 14.06.2001

Aufgabe 1: Sortieren mit Listenstrukturen

Entwickeln Sie ein Programm, das eine nicht bestimmte Anzahl Zahlen vom Typ Float einliest und dann entweder aufsteigend oder absteigend sortiert ausgibt. Der Benutzer soll die Folge von Zahlen nach einander eingeben. Nach der Eingabe der Zahlen gibt der Benutzer mit "ab" oder "auf" an, ob ab- oder aufsteigend sortiert werden soll.
Im letzten Praktikumsblatt haben wir unterschiedliche Sortierverfahren mit Arrays kennen gelernt. Arrays haben den Vorteil, dass auf die einzelnen Elemente mit einem Index zugegriffen werden kann, was die Zeit für den Zugriff auf einzelne Element minimal hält. Der Nachteil ist, dass ein Array, wenn es einmal definiert wurde, nur eine begrenzte Anzahl Elemente aufnehmen kann.
Mit einer einfach verketteten Liste ist die Anzahl der Elemente flexibel, aber der Zugriff auf ein Element dauert länger, da die Liste von Anfang an durchlaufen werden muss, bis das entsprechende Element gefunden wurde. Die Liste kann schon während der Eingabe sortiert werden.
Welche Möglichkeiten haben wir, mit Listen zu arbeiten?

1. Eine neue Liste erzeugen:

Eine Liste braucht einen Listenkopf, der die wichtigsten Informationen über die Liste enthält. Er enthält drei Verweiskomponenten, die auf das erste, das letzte und das aktuelle Element der Liste verweisen. Die Struktur eines einzelnen Elementes der Liste muss die Datenstruktur angeben, die ein Element enthalten soll, und mindestens eine Verweiskomponente enthalten, die auf das nächste Element verweist. Im Falle der doppelt verketteten Liste enthält jedes Listenelement noch eine Verweiskomponente, die auf das Vorgängerelement verweist. Abbildung 1 zeigt die Struktur des Listenkopfes und der Listenelemente für die einfach verkettete

Liste.
Mit der Definition der Struktur des Listenelements und des Listenkopfes ist die Liste als Datentyp festgelegt. Eine leere Liste wird mit der Definition einer Variable des Typs Listenkopf erzeugt.

```
TYPE ListenElementType;
TYPE ListenElementAccessType is access ListenElementType;
TYPE ListenElementType IS RECORD
  Wert: <Typ>;
  VerweisNachfolger: ListenElementAccessType;
END RECORD;
TYPE ListenKopfType IS RECORD
  VerweisListenAnfang: ListenElememtAccessType;
  VerweisListenEnde: ListenElementAccessType;
  VerweisAktuellesElement: ListenElementAccessType;
END RECORD;

ListenKopf: ListenKopfType;
```

2. Ein neues Element in eine Liste einfügen

Wir haben also jetzt mit dem Listenkopf eine neue Liste definiert. Die Verweisvariablen verweisen auf kein Element. Ihre Werte sind NULL. Durch die Allokation kann der Variablen VerweisListenAnfang ein Verweis auf ein neues Element zugewiesen werden.

```
ListenKopf.VerweisListenAnfang     := NEW ListenElementTyp(<Wert>, NULL);
ListenKopf.VerweisListenEnde       := ListenKopf.VerweisListenAnfang;
ListenKopf.VerweisAktuellesElement := ListenKopf.VerweisListenAnfang;
```

Dieses Element ist dann das erste, das letzte und das aktuelle Element der Liste, deshalb müssen auch die restlichen Verweisvariablen auf dieses Element verweisen.
Hat eine Liste n Elemente, so kann ein neues Element an n+1 Stellen eingefügt werden. Ein neues Element kann nach dem aktuellen Element, nach dem letzten Element oder vor das erste Element eingefügt werden. Immer vorausgesetzt, die Liste enthält bereits ein Element.

```
-- ListenKopf.VerweisAktuellesElement /= ListenKopf.VerweisListenEnde
ListenKopf.VerweisAktuellesElement.all.VerweisNachfolger :=
    NEW ListenElementType'(<Wert>,
                           ListenKopf.VerweisAktuellesElement.
                           all.VerweisNachfolger);

ListenKopf.VerweisListenEnde.all.VerweisNachfolger :=
    NEW ListenElementType'(<Wert>, NULL);
ListenKopf.VerweisListenEnde :=
    ListenKopfVerweisListenEnde.all.VerweisNachfolger;

ListenKopf.VerweisListenAnfang :=
    NEW ListenElementType(<Wert>, ListenKopf.VerweisListenAnfang);
```

Abbildung 2 zeigt, wie ein neues Element in eine einfach verkettete Liste angehängt oder eingefügt wird.

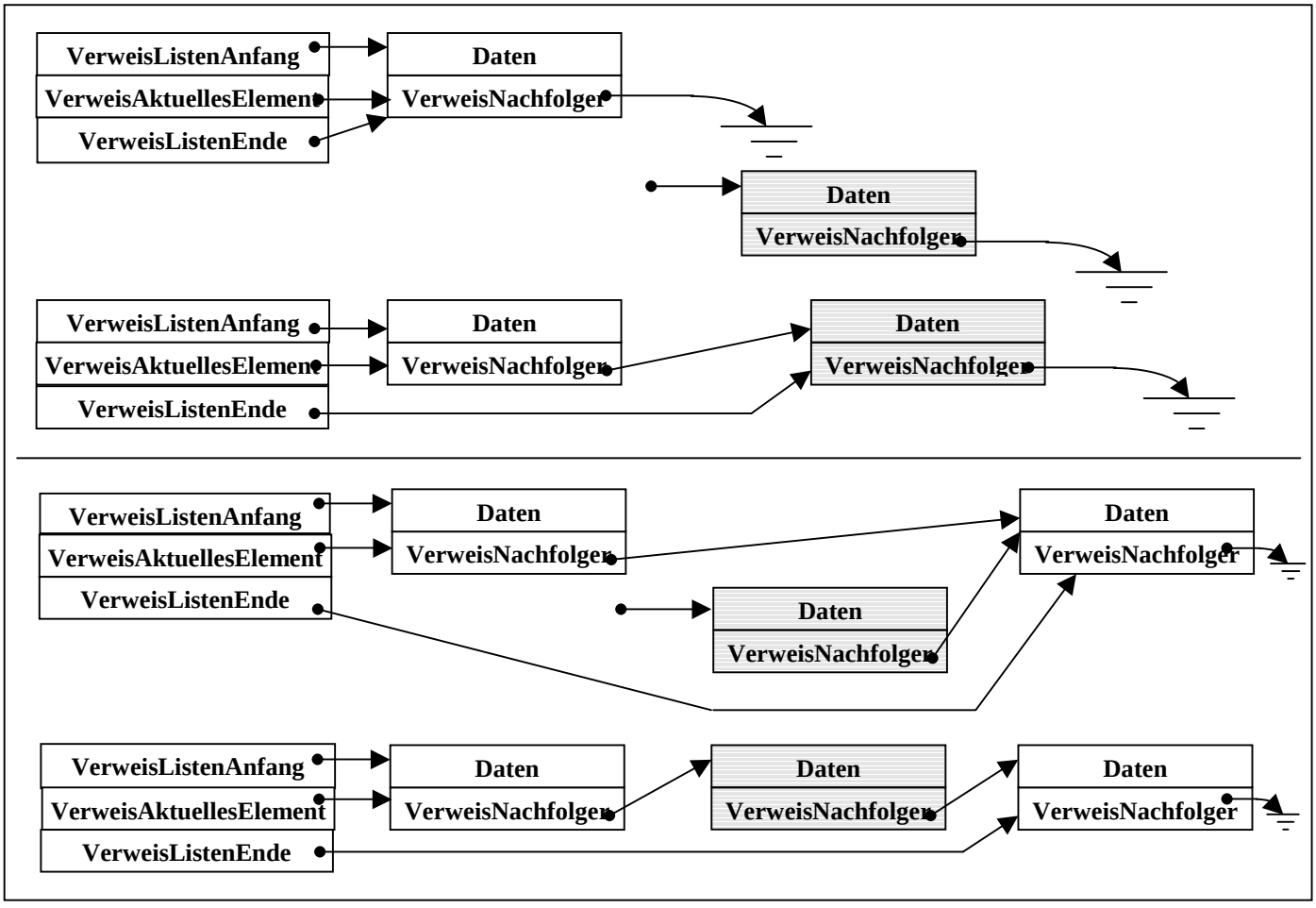

Abbildung 2: anhängen oder einfügen eines neuen Elements in eine Liste

3. Ein Element aus der Liste entfernen

Es müssen wieder einige Fälle unterschieden werden. Aus einer leeren Liste kann kein Element entfernt werden, wir nehmen also an, dass die Liste nicht leer ist. Weitere Fälle sind das erste und das letzte Element. Wird das erste Element entfernt, so muss überprüft werden, ob das aktuelle Element auf das zu entfernende Element verweist und ob das zu entfernende das letzte Element ist. Für das Entfernen des letzten Element muss überprüft werden ob das letzte Element auch gleichzeitig das erste Element ist. Zuletzt, um den Nachfolger des aktuelle Elementes zu entfernen muss überprüft werden, ob das aktuelle Element einen Nachfolger hat und ob dieser Nachfolger gleichzeitig das letzte Element ist. Folgender Codeausschnitte realisiert alle möglichen Fälle, solange die Liste nicht leer ist.

```
-- Entfernen des ersten Elementes
IF ListenKopf.VerweisListenAnfang = ListenKopf.VerweisAktuellesElement
THEN ListenKopf.VerweisAktuellesElement :=
         ListenKopf.VerweisAktuellesElement.all.VerweisNachfolger;
  -- Das aktuelle Element verweist auf das zu entfernende erste Element
END IF;
IF ListenKopf.VerweisListenAnfang = ListenKopf.VerweisListenEnde
THEN ListenKopf.VerweisListenEnde := NULL;
  -- Die Liste besteht aus einem Element;
END IF;
ListenKopf.VerweisListenAnfang :=
      ListenKopf.VerweisListenAnfang.all.VerweisNachfolger;

-- Entfernen des letzten Elementes
IF ListenKopf.VerweisListenAnfang=ListenKopf.VerweisListenEnde THEN
  -- Die Liste besteht aus einem Element
  ListenKopf.VerweisListenAnfang     := NULL;
  ListenKopf.VerweisListenEnde       := NULL;
  ListenKopf.VerweisAktuellesElement := NULL;
ELSE
  -- Das aktuelle Element muss zum Entfernen des letzten Elementes auf
  -- das vorletzte verweisen
  ListenKopf.VerweisAktuellesElement := ListenKopf.VerweisListenAnfang;
  WHILE ListenKopf.VerweisAktuellesElement.all.VerweisNachfolger/=
        ListenKopf.VerweisListenEnde LOOP
    ListenKopf.VerweisAktuellesElement :=
          ListenKopf.VerweisAktuellesElement.all.VerweisNachfolger;
  END LOOP;
  ListenKopf.VerweisListenEnde:=ListenKopf.VerweisAktuellesElement;
  ListenKopf.VerweisListenEnde.all.VerweisNachfolger := NULL;
END IF;

-- Entfernen des Nachfolgers des aktuellen Elementes;
IF ListenKopf.VerweisAktuellesElement.all.VerweisNachfolger = NULL THEN
  NULL; --  Das aktuelle Element hat keinen Nachfolger zum Entfernen
ELSIF ListenKopf.VerweisAktuellesElement.all.VerweisNachfolger =
      ListenKopf.VerweisListenEnde THEN
  -- Das aktuelle Element ist das vorletzte Element und das letzte
  -- Element ist zu entfernen
  ListenKopf.VerweisListenEnde := ListenKopf.VerweisAktuellesElement;
  ListenKopf.VerweisListenEnde.all.VerweisNachfolger := NULL;
ELSE
  -- Es wird ein Element der Liste entfernt, das nicht das Letzte ist
  ListenKopf.VerweisAktuellesElement.all.VerweisNachfolger :=
        ListenKopf.VerweisAktuellesElement.
        all.VerweisNachfolger.all.VerweisNachfolger;
END IF;
```

Abbildung 3 zeigt das Entfernen eines Elements aus einer Liste. Auf das aus der Liste entfernte Element kann nicht mehr zugegriffen werden, wenn keine andere Verweisvariable mehr darauf verweist.

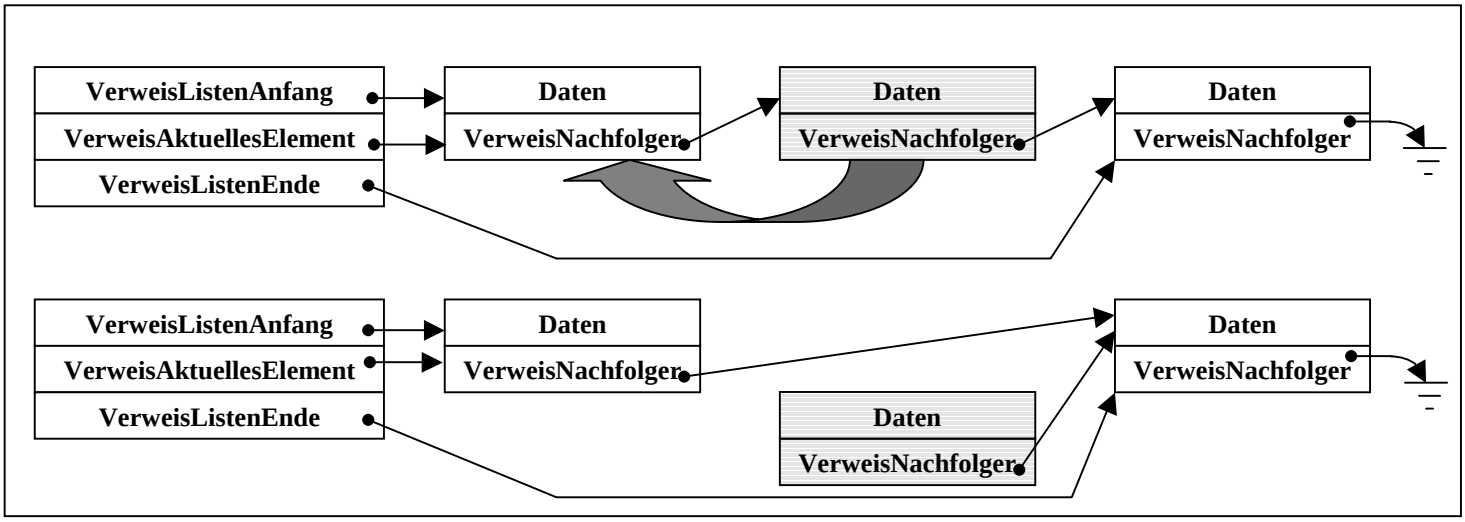

Abbildung 3: Das Entfernen eines Elements aus einer Liste