Lehrstuhl für Informatik 3
Rheinisch-Westfälische Technische Hochschule
Aachen
Univ.-Prof. Dr.-Ing. M. Nagl

Aachen, 16. 06. 2006

Nagl, Becker, Ranger, Wörzberger

Übungen zur Vorlesung „Grundgebiete der Informatik 2: Algorithmen und Programmiertechniken“

— Blatt 9 —

20. Aufgabe *Anwendungen von Listen*: *(5 Punkte)*

Sie haben jetzt einige abstrakte Datenstrukturen und deren Implementierungen kennengelernt. So kann eine Liste (sequentielle Struktur) als Feld, als Kette von Elementen mit Zeigern oder als Ringliste abgelegt werden. Große Listen wird man gleich in Dateien auslagern wollen, winzige Listen direkt in Hardware implementieren. In dieser Aufgabe sollen sie dieses Strukturwissen anwenden und durch eigenes Nachdenken erweitern.

(a) Nennen Sie fünf Anwendungsbeispiele für sequentielle Strukturen aus der Elektrotechnik. Fassen Sie ‘sequentielle Strukturen’ und ‘Elektrotechnik’ ruhig weit auf. *(3 Punkte)*

(b) Nennen Sie für jedes ihrer Anwendungsbeispiele eine geeignete Implementierung. Sie sind auch hier nicht auf die Beispiele aus der Vorlesung eingeschränkt. *(2 Punkte)*

21. Aufgabe *Doppelt verkettete Liste*: *(10 Punkte)*

In der Vorlesung wurde ein ADT (abstrakter Datentyp) für eine Liste vorgestellt. Bei der Effizienzbetrachtung stellte sich heraus, das eine doppelt verkettete Liste in einigen Punkten effizienter ist und vor allem zusätzliche Möglichkeiten bietet.

In dieser Aufgabe sollen Sie eine Klasse für einen ADT “Doppelt verkettete Liste” mit der unten angegebenen Schnittstelle implementieren.

(a) Die Datentypdeklaration für ein Listenelement (`struct ListItem`) ist opak. Formulieren Sie diese aus. *(2 Punkte)*

(b) Implementieren Sie die einzelnen Methoden des ADT “Doppelt verkettete Liste” (Klasse `DLList`). Achten Sie hierbei vor allem darauf, daß sich die Methoden aufeinander abstützen, daß heißt die Methode `readNext` sollte beispielsweise unter anderem mit Hilfe der Methode `next` implementiert werden. *(8 Punkte)*

```
/* ADT Doppelt verkettete Liste (Double Linked List) -- interface */
#ifndef __INC_DLList_H__
#define __INC_DLList_H__

class DLList {
public:
    /* Erzeugt leere Liste. */
```

```
    DLList();

    /* Haengt ein Element hinten an die Liste an. */
    void append(int item);

    /* Loescht das Element, auf dem der Cursor steht. */
    void deleteCurrent();

    /* Loescht das Element an der angegebenen Stelle. */
    void deleteIndex(int index);

    /* Bewegt den Cursor an den Anfang der Liste. */
    void toStart();

    /* Bewegt den Cursor an das Ende der Liste. */
    void toEnd();

    /** Bewegt den Cursor zum naechsten Element in der Liste. */
    void next();

    /** Bewegt den Cursor zum vorherigen Element in der Liste. */
    void prev();

    /* Liest das aktuelle Element in der Liste. */
    int read();

    /* Liest das aktuelle Element in der Liste und bewegt den Cursor eins weiter. */
    int readNext();

    /* Liest das aktuelle Element in der Liste und bewegt den Cursor eins zurueck. */
    int readPrev();

    /* Liefert true zurueck, wenn die Liste nicht leer ist. */
    bool isNonEmpty();

    /* Liefert true zurueck, wenn die Liste nicht voll ist. */
    bool isNonFull();

    /* Liefert true zurueck, wenn der Cursor am Anfang der Liste steht. */
    bool isAtEnd();

    /* Liefert true zurueck, wenn der Cursor am Ende der Liste steht. */
    bool isAtStart();

private:
    struct ListItem;          // Opake Definition eines Listenelements

    ListItem *first;          // Zeiger auf das erste Element der Liste
    ListItem *last;           // Zeiger auf das letzte Element der Liste
    ListItem *current;        // Zeiger auf das aktuelle Element (=Cursor)

    bool nonFull;             // true, wenn Liste nicht voll
    bool nonEmpty;            // true, wenn Liste nicht leer
};

#endif
```

22. Aufgabe *Bäume ausgeben*: *(5 Punkte)*

Implementieren sie den Rumpf des funktionalen Moduls `BinBaumAusgabe`, dessen Schnittstelle unten angegeben ist. Benutzen Sie dazu den Datentypmodul `BinBaum`, von dem unten nur die Schnittstelle angegeben ist; die Implementation des Moduls finden Sie wieder einmal bei `http://www-i3.informatik.rwth-aachen.de/ggdi`.

Modul `BinBaumAusgabe`

```
/* Funktionaler Modul BinBaumAusgabe --- Ausgabe von Baeumen des adt BinBaum

   Die Funktionen in diesem Modul erzeugen verschiedene Ausgaben eines
   (Teil-)Baums des vom Modul BinBaum exportierten Typs BinBaumKnotenT.
*/

#ifndef __INC_BinBaumAusgabe_h__
#define __INC_BinBaumAusgabe_h__
#include "BinBaum.h"            // benutzt: Navigation und Wert auslesen
#include <iostream>           // benutzt: ostream, cout

using namespace std;

/* Gibt den durch den Knoten angezeigten (Teil-)Baum mit einem
   Inorder-Durchlauf aus, mit je einem Leerzeichen Einrueckung pro
   Schachtelung.
Vorbedingung:  knoten ein gueltiger Baumknoten
Eingabe:       knoten  --- die Wurzel des auszugebenden (Teil-)Baums
               ausgabe --- Ausgabe-Datenstrom
Ausgabe:       Darstellung auf ausgabe
Nachbedingung: (Teil-)Baum von knoten unveraendert
*/
void gebeBaumInorderAus(BinBaumKnotenT const & knoten, ostream& ausgabe=cout);
///
#endif
```

Modul `BinBaum`

```
/* Datentypmodul BinBaum --- Binaerer Baum
   Baeume dieses Datentyps speichern vorzeichenbehaftete ganze Zahlen.  Die
   verkapselte Entwurfsentscheidung ist die interne Speicherung der Knoten.
   Die Namen befolgen die Baum-Metapher so weit wie moeglich.
   Die Kommentare sind aus Platzgruenden sparsamer.
*/
#ifndef __INC_BinBaum_h__
#define __INC_BinBaum_h__

struct BinBaumKnotenT;          // Der opake Eintragstyp.

/* Erzeugungs- und Loeschoperationen */
BinBaumKnotenT & neuerKnoten();
void zerstoereKnoten(BinBaumKnotenT & opfer);

/* Zugriff auf den gespeicherten Wert */
int  leseWert (BinBaumKnotenT const & knoten);
void setzeWert(BinBaumKnotenT       & knoten, int neuerWert);

/* Navigation zu den Nachbarn.  Liefern NULL und geben eine Fehlermeldung
   aus, falls linkerAstExistiert (etc.) false liefern. */
BinBaumKnotenT * linkerAst (BinBaumKnotenT const & stamm);
BinBaumKnotenT * rechterAst(BinBaumKnotenT const & stamm);
```

```
BinBaumKnotenT * stamm     (BinBaumKnotenT const & ast  );

/* Abfragen zur Struktur */
bool linkerAstExistiert (BinBaumKnotenT const & knoten);
bool rechterAstExistiert(BinBaumKnotenT const & knoten);
bool stammExistiert     (BinBaumKnotenT const & knoten);

/* Struktur des Baumes aendern: Aeste hinzufuegen oder absaegen.  Der Stamm
   bleibt.  */
/* Neue Aeste anbringen.  Eventuell vorhandene Aeste werden dabei abgesaegt,
   aber nicht zerstoert. */
void setzeLinkenAst (BinBaumKnotenT & stammKnoten,
                     BinBaumKnotenT & neuerLinkerAstKnoten);
void setzeRechtenAst(BinBaumKnotenT & stammKnoten,
                     BinBaumKnotenT & neuerRechterAstKnoten);

/* Die Aeste werden abgesaegt, aber nicht zerstoert. */
void loescheLinkenAst (BinBaumKnotenT & stammKnoten);
void loescheRechtenAst(BinBaumKnotenT & stammKnoten);
#endif
```

Abgabe: 23. 06. 2006 vor der Vorlesung.