Doppelt verkettete Listen: Das Modul dclists

Das Modul `dclists` stellt Datentypen und Operationen zur Verwaltung und Bearbeitung doppelt verketteter Listen (doubly connected lists) zur Verfügung. Abschnitt 1 enthält allgemeine Erläuterungen zu doppelt verketteten Listen. Abschnitt 2 behandelt die definierten Konstanten und Abschnitt 3 die verwendeten Datentypen. Abschnitt 4 gibt eine Übersicht über die Funktionen, die auf doppelt verkettete Listen angewandt werden können. Die Abschnitte 5 und 6 beschreiben zwei mögliche Implementierungen doppelt verketteter Listen, und zwar einmal als echte Listen und einmal als Vektoren.

Das Modul `dclists` besteht aus der Datei `dclists.h`, die die Deklarationen der Datentypen und der Funktionen enthält, mit denen doppelt verkettete Listen bearbeitet werden können. Diese Datei ist mit `#include "dclists.h"` in die Datei einzubinden, in der doppelt verkettete Listen verwendet werden sollen. Implementiert sind die Funktionen in den Dateien `dclists.c` (echte Listen) und `dcl_vec.c` (Vektoren). Nur eine der beiden Dateien, entweder `dclists.c` oder `dcl_vec.c`, wird benötigt.

Das Modul `dclists` benutzt nur Funktionen der Standardbibliotheken und ist daher unabhängig von anderen Modulen.

1 Allgemeine Erläuterungen

Eine doppelt verkettete Liste ist eine Struktur, in der Daten linear angeordnet sind. Das bedeutet, daß es zu jedem Element (außer dem letzten der Liste) genau einen Nachfolger und zu jedem Element (außer dem ersten der Liste) genau einen Vorgänger gibt. Diese Nachfolgerbeziehung definiert die Struktur einer Liste. Sie heißt doppelt verkettet, wenn sowohl der Nachfolger als auch der Vorgänger eines Elementes leicht (d.h. effizient) bestimmt werden können. Zu einfach verketteten Listen, in denen nur der Nachfolger eines eines Elementes leicht (d.h. effizient) bestimmt werden kann, sehe man das Modul `sclists` (singly connected lists).

Auf einer doppelt verketteten Liste sind mindestens folgende Operationen möglich:

- Anlegen und Löschen einer Liste
- Einfügen und Entfernen eines Elementes
- Auslesen eines Elementes
- Aufsuchen des Vorgängers oder des Nachfolgers eines Elementes

Weitere Operationen können das Suchen in einer Liste und das Sortieren der Listenelemente sein (s.u.). Die in dem hier beschriebenen Modul `dclists` zur Verfügung gestellten Operationen gehen weit über die Grundoperationen doppelt verketteter Listen hinaus. Sie sind in der Datei `dclists.h` definiert (die (s.o.) in das Programm, in dem doppelt verkettete Listen benutzt werden sollen, eingebunden werden muß) und in Abschnitt 4 beschrieben.

Die in der Datei `dclists.h` deklarierten Funktionen sind unabhängig von der Implementierung der Liste (ob als echte Liste oder als Vektor — deshalb kann auch für die beiden Implementierungen in `dclists.c` und `dcl_vec.c` die gleiche Header-Datei `dclists.h` verwendet werden) und außerdem von den in der Liste zu speichernden Daten (es kann sich um ganze Zahlen, Fließkommazahlen, Zeichenketten oder Strukturtypen handeln). Sie schreibt nicht einmal vor, daß die in den Elementen gespeicherten Daten alle die gleiche Größe (gemessen in der Menge des benötigen Speichers) haben müssen. Jedem Datenbereich kann eine individuelle Größe gegeben werden (vgl. die Beschreibung der Funktion `dcl_insert`).

Listencursor

Das Einfügen, Auslesen, Entfernen eines Elementes und das Aufsuchen des Nachfolgers eines Elementes beziehen sich auf eine bestimmte Position in der linearen Anordnung der Elemente. Z.B. muß angegeben werden, an welche Stelle der Ordnung ein neues Element eingefügt werden soll (etwa: vor welchem bereits vorhandenen Element) oder welches Element (das Element an welcher Position) gelöscht werden soll. Um solche Positionen festlegen zu können, ist es günstig, einen sogenannten Cursor, d.i. ein Verweis auf das aktuelle Element der Liste, zu verwalten. Dieser Cursor gibt die Position in der Liste an, auf die sich die Operationen beziehen, die eine Positionsangabe benötigen. Er zeigt z.B. auf das Element, das entfernt werden soll.

Ein Listencursor ist analog zu einem Cursor in einem Texteditor. Ein Cursor in einem Texteditor steht stets in der Zeile, auf die sich Operationen beziehen, die z.B. mit Hilfe von Steuertasten über die Tastatur eingegeben werden. Soll etwa eine Zeile gelöscht werden, so wird die Zeile gelöscht, in der der Cursor steht.

Die Verwendung eines Cursors hat den großen Vorteil, daß sich Operationen wie das Löschen und das Einfügen von Elementen vom Aufsuchen des zu löschenden Elementes bzw. des Elementes, vor oder nach dem eingefügt werden soll, trennen lassen. Dadurch lassen sich viele Operationen einfacher und verständlicher gestalten. Man stelle sich z.B. vor, man müßte in einem Texteditor erst auf andere Weise die zu löschende Zeile festlegen, etwa durch Eintippen des Zeilenanfangs.

Wird ein Cursor verwendet, um die Einfügeposition in einer Liste zu bestimmen, so gibt es zwei Möglichkeiten: entweder das neue Element wird vor oder hinter dem aktuellen Element eingefügt. In der vorliegenden Implementierung wurde ein Einfügen *vor* dem aktuellen Element gewählt, da die Einfügeposition oft durch eine Suche bestimmt wird. Diese liefert aber meist das erste Element, auf das ein bestimmtes Kriterium *nicht* mehr zutrifft, und *vor* dem dann ein neue Element eingefügt werden muß. Sollen z.B. ganze Zahlen aufsteigend in eine Liste eingefügt werden, so liefert eine Suche nach der Einfügestelle für eine neue Zahl das erste Element, für das die enthaltene Zahl *nicht* mehr kleiner ist als die einzufügende Zahl, und *vor* der folglich das neue Element eingefügt werden muß.

2 Konstanten

Zur Bezeichnung der Position, auf die der Listencursor mit Hilfe der Funktion `dcl_reset` zurückgesetzt, und der Richtung, in die der Listencursor mit Hilfe der Funktion `dcl_get` bewegt werden soll, sind die folgenden Konstanten definiert (vgl. Abschnitt 4:

```
/* --- parameter 'pos' of function dcl_reset() -- */
#define DCL_FRONT    -1         /* in front of first element */
#define DCL_FIRST     0         /* on first element */
#define DCL_LAST      1         /* on last  element */
#define DCL_BACK      2         /* behind last element */

/* --- parameter 'move' of function dcl_get() --- */
#define DCL_FORWARD   1         /* go to successor element */
#define DCL_BACKWARD -1         /* go to predecessor element */
#define DCL_STAY      0         /* stay on current element */
```

Die Angabe des Wertes `DCL_FRONT` für den Parameter `pos` der Funktion `dcl_reset` setzt den Listencursor vor das erste Listenelement, die Angabe des Wertes `DCL_FIRST` auf das erste Listenelement. Im ersten Fall muß der Cursor erst auf das nächste Element bewegt werden (mit Hilfe der Funktion `dcl_get`), ehe das erste Listenelement ausgelesen werden kann. Die Angabe des Wertes `DCL_BACK` setzt den Listencursor hinter das letzte Listenelement, die Angabe des Wertes `DCL_LAST` auf das letzte Listenelement. Im ersten Fall muß der Listencursor erst auf das vorhergehende Listenelement bewegt werden (mit Hilfe der Funktion `dcl_get`), ehe das letzte Listenelement ausgelesen werden kann.

Die Angabe des Wertes `DCL_FORWARD` für den Parameter `move` der Funktion `dcl_get` bewirkt, daß der Listencursor erst auf das nächste Listenelement bewegt wird, ehe das Element unter dem Listencursor (das neue aktuelle Element) ausgelesen wird. Die Angabe des Wertes `DCL_BACKWARD` bewirkt, daß der Listencursor erst auf das vorhergehende Element bewegt wird. Durch den Wert `DCL_STAY` wird angegeben, daß der Listencursor nicht bewegt, sondern an seiner aktuellen Position verbleiben soll.

3 Datentypen

Die Datei `dclists.h` deklariert den Strukturdatentyp `DCLISTS` zur Darstellung doppelt verketteter Listen. Die Strukturdefinition ist jedoch verborgen (sie unterscheidet sich je nach Implementierung und steht in der Datei `dclists.c` bzw. der Datei `dcl_vec.c`), so daß von außen (d.h. aus einem Programm, in dem doppelt verkettete Listen verwendet werden) nur über die deklarierten Funktionen Änderungen an einer Instanz dieser Struktur vorgenommen werden können. Die Datei `dclists.h` definiert also einen abstrakten Datentyp zusammen mit einer Menge auf diesem Typ definierter Operationen, durch die allein Instanzen dieses Typs bearbeitet werden können.

Einige der in Datei `dclists.h` definierten Funktionen erhalten Funktionen als Parameter, die auf die in den Listenelementen gespeicherten Daten angewendet werden können. Um die Deklarationen dieser Funktionen zu vereinfachen, werden drei Funktionstypen definiert, und zwar die Typen `DCL_DELFN` (doubly connected list deletion function), `DCL_APPFN` (doubly connected list application function) und `DCL_CMPFN` (doubly connected list comparison function).

Der erste Funktionstyp ist definiert als

```
typedef void DCL_DELFN (void *p);
```

Es handelt sich um eine ergebnislose Funktion, die einen typenlosen Zeiger als Parameter erhält. Sie wird mit einem Zeiger auf den Datenbereich eines Listenelementes aufgerufen. Falls in den Datenbereichen Objekte abgelegt sind, für die Speicher angefordert wurde, sollte diese Funktion diesen Speicher wieder freigeben.

Der zweite Funktionstyp ist definiert als

```
typedef void DCL_APPFN (void *call, void *client);
```

Es handelt sich um eine ergebnislose Funktion, die zwei typenlose Zeiger als Parameter erhält. Sie wird mit einem Zeiger auf den Datenbereich eines Listenelementes (`call`) und einem Zeiger auf einen zusätzlichen, vom Programmierer anzugebenden Datenbereich (`client`) aufgerufen. Letzterer Zeiger kann beim Aufruf der Funktion `dcl_apply` übergeben werden. Was die Funktion mit diesen Datenbereichen tut, liegt in der Verantwortung des Programmierers, der einfach verkettete Listen benutzt. Es kann sich z.B. um eine Funktion handeln, die die in dem Datenbereich eines Listenelementes gespeicherten Daten auf dem Bildschirm ausgibt. Übergibt man eine solche Ausgabefunktion als zweiten Parameter an die Funktion `dcl_apply` (siehe Abschnitt 4), so wird sie auf jedes Element der (als erster Parameter angegebenen) Liste angewandt und damit die gesamte Liste Element für Element ausgegeben. Der Parameter `client` kann in diesem Fall ignoriert werden oder zur Übergabe einer Beschreibung des Ausgabeformats dienen.

Der dritte Funktionstyp ist definiert als

```
typedef int DCL_CMPFN (const void *p1, const void *p2);
```

Diese Funktion darf die Datenbereiche der Listenelemente, auf die sie Zeiger als Parameter erhält, nicht verändern (wegen des Typmodifizierers `const`), d.h. sie darf sie nur lesen. Aus den in den Datenbereichen abgelegten Daten hat sie, eventuelle unter Verwendung der über den Zeiger `client` übergebenen, der Verantwortung des Programmierers unterliegenden Daten (vgl. den zweiten Parameter des Funktionstyps `DCL_APPFN`), eine ganze Zahl zu bestimmen. Diese Zahl sollt das Ergebnis eines Vergleich dieser Daten wiedergeben, denn sie z.B. wird als Parameter der Funktion `dcl_sort` (siehe Abschnitt 4) zum Sortieren einfach verketteter Listen verwendet. Sind die Daten z.B. ganze Zahlen, so könnte sie das Größenverhältnis der beiden Zahlen angeben, indem sie einen Wert größer als 0 zurückliefert, wenn die erste Zahl größer ist als die zweite, einen Wert kleiner 0, wenn die erste Zahl kleiner ist als die zweite, und den Wert 0, wenn beide Zahlen gleich sind.[1]

4 Funktionen

In diesem Abschnitt werden die Funktionen vorgestellt, mit denen doppelt verkettet Listen angelegt, bearbeitet und wieder gelöscht werden können. Alle aufgeführten Funktionen sind in der Datei `dclists.h` deklariert. Da doppelt verkettete Listen als abstrakter Datentyp ausgelegt sind, sollten sie nur über die in diesem Abschnitt angegebenen Funktionen verändert werden. In der objektorientierten Sprechweise sind die hier aufgeführten Funktionen die Methoden, die auf ein Objekt der Klasse ‘doppelt verkettete Liste’ angewendet werden können.

`dcl_create`

Name	“create a doubly connected list” — Erzeugen einer doppelt verketteten Liste
Definition	`DCLIST* dcl_create (DCL_DELFN delfn);`
Beschreibung	Es wird eine doppelt verkettete Liste angelegt, die zunächst leer ist, d.h. keine Elemente enthält. Der Cursor der Liste steht hinter dem letzten Element der Liste. Die Funktion `delfn` wird in der Liste gemerkt und beim Löschen eines Listenelementes (durch eine der Funktionen `dcl_delete` oder `dcl_remove`) mit einem Zeiger auf seinen Datenbereich als Parameter aufgerufen. Dies geschieht, um dem aufrufenden Programm die Möglichkeit zu geben, in diesem Datenbereich abgelegte Objekte, für die Speicher angefordert wurde, freizugeben. Ist dies nicht nötig, so kann für `delfn` der Wert `(DCL_DELFN*)0` übergeben werden.

[1] Man beachte, daß die einfache Bildung der Differenz der beiden Zahlen zwar das geforderte Verhalten zeigt, es wegen der Möglichkeit von Zahlenüberläufen jedoch angebrachter ist, durch Vergleiche nur das Vorzeichen der Differenz zu bestimmen.

Ergebnis	Falls die Liste angelegt werden konnte, wird ein Zeiger auf diese Liste, anderenfalls `NULL` zurückgegeben.

`dcl_delete`

Name	"delete a doubly connected list" — Löschen einer doppelt verketteten Liste
Definition	`void dcl_delete (DCLIST *list);`
Beschreibung	Die doppelt verkettete Liste `list` und alle in ihr enthaltenen Elemente werden gelöscht. Für jedes Listenelement wird vor dem Löschen die beim Anlegen der Liste übergebene Funktion `delfn` mit einem Zeiger auf seinen Datenbereich als Parameter aufgerufen (siehe Funktion `dcl_create`).
Ergebnis	Die Funktion hat kein Ergebnis.

`dcl_len`

Name	"get the length of a doubly connected list" — Bestimmen der Länge einer doppelt verketteten Liste
Definition	`int dcl_len (DCLIST *list);`
Beschreibung	Es wird die Länge der doppelt verketteten Liste `list`, d.h. die Zahl der in ihr enthaltenen Elemente, bestimmt.
Ergebnis	Die Länge der doppelt verketteten Liste `list`.

`dcl_reset`

Name	"reset the list cursor" — Rücksetzen des Cursors der Liste
Definition	`void* dcl_reset (DCLIST *list, int pos);`
Beschreibung	Hat der Parameter `pos` den Wert `DCL_FRONT`, so wird der Cursor der doppelt verketteten Liste `list`, d.i. der Zeiger auf das aktuelle Element der Liste, vor das erste Element der Liste `list` gesetzt, hat `pos` den Wert `DCL_FIRST`, so wird er auf das erste Element der Liste `list` gesetzt. Hat der Parameter `pos` den Wert `DCL_BACK`, so wird der Listencursor hinter das letzte Element der Liste `list` gesetzt, hat `pos` den Wert `DCL_LAST`, so wird er auf das letzte Element der Liste `list` gesetzt. Ist die Liste leer und sollte der Cursor auf das erste Element der Liste gesetzt werden, so wird er vor das erste Element, sollte er auf das letzte, so wird er hinter das letzte Element gesetzt. Steht der Cursor nach dem Setzen auf seine neue Position auf einem Element der Liste, so wird anschließend der Datenbereich dieses Elementes ausgelesen.
Ergebnis	Falls der Listencursor nach dem Setzen auf einem Element der Liste steht, so wird ein Zeiger auf den Datenbereich dieses Elementes, anderenfalls `NULL` zurückgegeben.

`dcl_insert`

Name	"insert into a doubly connected list" — Einfügen in eine doppelt verkettete Liste
Definition	`void* dcl_insert (DCLIST *list, size_t size);`

Beschreibung In die doppelt verkettete Liste `list` wird vor dem Cursor der Liste, d.h. vor dem aktuellen Element, ein neues Listenelement mit einem Datenbereich der Größe `size` eingefügt. Dieser Datenbereich wird *nicht* initialisiert. Der Cursor der Liste ändert seine Position nicht: nach dem Einfügen zeigt er auf das gleiche Element wie vorher. Dies gilt auch, wenn er hinter dem letzten Element der Liste stand: Das eingefügte Listenelement ist dann das letzte der Liste und der Cursor steht weiterhin hinter dem (wenn auch jetzt neuen) letzten Element der Liste.

Ergebnis Falls ein Listenelement eingefügt werden konnte, wird ein Zeiger auf den Datenbereich dieses Elementes, anderenfalls `NULL` zurückgegeben.

`dcl_get`

Name "get a datum from a doubly connected list" — Lesen eines Datums aus einer doppelt verketteten Liste

Definition `void* dcl_get (DCLIST *list, int move);`

Beschreibung Falls der Parameter `move` den Wert `DCL_STAY` hat, so bleibt die Position des Cursors der doppelt verketteten Liste `list` unverändert. Hat `move` den Wert `DCL_FORWARD`, so wird der Listencursor auf das Element gesetzt, das auf das aktuelle Element folgt. Ist das aktuelle Element das letzte der Liste, so steht er anschließend hinter dem letzten Element der Liste (und damit auf keinem Element). Hat `move` den Wert `DCL_BACKWARD`, so wird der Listencursor auf das Element gesetzt, das dem aktuellen Element vorausgeht. Ist das aktuelle Element das erste der Liste, so steht er anschließend vor dem ersten Element der Liste (und damit auf keinem Element). Steht der Cursor nach dem Versetzen nicht vor dem ersten oder hinter dem letzten Element der Liste, so wird der Datenbereich desjenigen Listenelementes ausgelesen, auf das der Cursor zeigt.

Ergebnis Falls es (nach einem eventuellen Versetzen des Cursors) ein aktuelles Element gibt, so wird ein Zeiger auf den Datenbereich dieses Elementes, anderenfalls, d.h. wenn der Cursor der Liste (nach einem eventuellen Versetzen) vor dem ersten oder hinter dem letzten Element steht, so wird `NULL` zurückgegeben.

`dcl_remove`

Name "remove from a doubly connected list" — Löschen aus einer doppelt verketteten Liste

Definition `void dcl_remove (DCLIST *list, int all);`

Beschreibung Falls der Parameter `all` den Wert 0≙falsch hat, so wird das aktuelle Element der doppelt verketteten Liste `list`, d.h. das Element, auf dem der Cursor der Liste steht, gelöscht. Vor dem Löschen wird die beim Anlegen der Liste übergebene Funktion `delfn` mit einem Zeiger auf den Datenbereich dieses Elementes als Parameter aufgerufen (siehe Funktion `dcl_create`). Der Cursor der Liste wird auf das Element gesetzt, das auf das gelöschte Element folgte. War das gelöschte Element das letzte der Liste, so steht er anschließend hinter dem letzten Element der Liste (und damit auf keinem Element).

Hat `all` einen Wert ungleich 0, d.h. den Wert wahr, so werden *alle* Elemente der Liste `list` gelöscht. Für jedes Listenelement wird vor dem Löschen die beim

Anlegen der Liste übergebene Funktion `delfn` mit einem Zeiger auf den Datenbereich des Elementes als Parameter aufgerufen (siehe Funktion `dcl_create`). Die Liste selbst bleibt erhalten. Nach dem Löschen ist sie jedoch leer, d.h. sie enthält keine Elemente. Der Cursor der Liste steht hinter dem letzten Element der leeren Liste.

Ergebnis: Die Funktion hat kein Ergebnis.

`dcl_search`

Name "search a datum in a doubly connected list" — Suchen eines Datums in einer doppelt verketteten Liste

Definition

```
void* dcl_search (DCLIST *list, const void *datum,
                  int dir, DCL_CMPFN cmpfn, void *client);
```

Beschreibung Beginnend mit den aktuellen Element der Liste `list`, d.h. dem Element auf dem der Cursor der Liste `list` steht, wird nach einem Element gesucht, dessen Datenbereich ein Datum enthält, das gemäß der Vergleichsfunktion `cmp` zu dem übergebenen Datum `datum` gleich ist. Dazu werden die Listenelemente durchlaufen und die Vergleichsfunktion `cmp` für jedes dieser Elemente mit einem Zeiger auf das übergebene Datum (erster Parameter), einem Zeiger auf den Datenbereich des Elementes (zweiter Parameter) und den Zeiger `client` (dritter Parameter) aufgerufen. Die Suchrichtung wird durch den Parameter `dir` angegeben. Hat er den Wert `DCL_FORWARD`, so wird vom aktuellen Element aus vorwärts (in Nachfolgerrichtung), hat er den Wert `DCL_BACKWARD` so wird rückwärts (in Vorgängerrichtung) gesucht. Die Suche wird abgebrochen, wenn entweder die Vergleichsfunktion für ein Element den Wert 0 zurückgibt oder das Ende der Liste erreicht ist. Der Cursor der Liste wird auf das gefundene Listenelement, oder, wenn kein passendes Element gefunden werden konnte, hinter das letzte Element bzw. vor das erste Element der Liste gesetzt.

Ergebnis Falls ein Element gefunden werden konnte, dessen Datenbereich ein zu dem übergebenen Datum gleiches Datum enthält, so wird ein Zeiger auf den Datenbereich dieses Elementes, anderenfalls der Wert `NULL` zurückgegeben.

`dcl_apply`

Name "apply a function to all list elements" — Anwenden einer Funktion auf alle Listenelemente

Definition

```
void dcl_apply (DCLIST *list, DCL_APPFN appfn, void *client);
```

Beschreibung Die Funktion `appfn` wird auf alle Elemente der doppelt verketteten Liste `list` angewandt (mit einem Zeiger auf ihren Datenbereich als Parameter aufgerufen). `client` ist ein Zeiger auf einen beliebigen Datenbereich, in dem Daten abgelegt sind, die für die Bearbeitung der Objekte gebraucht werden. Dieser Zeiger wird beim Aufruf der Funktion `appfn` als zweiter Parameter übergeben.

Ergebnis Die Funktion hat kein Ergebnis.

dcl_sort

Name	"sort a doubly connected list" — Sortieren einer doppelt verketteten Liste
Definition	`void dcl_sort (DCLIST *list, DCL_CMPFN cmpfn);`
Beschreibung	Die doppelt verkettete Liste `list` wird gemäß der Vergleichsfunktion `cmpfn` aufsteigend sortiert. Der Cursor der Liste wird zurückgesetzt (vgl. die Funktion `dcl_reset`). Die Vergleichsfunktion `cmpfn` wird mit Zeigern auf die Datenbereiche zweier Listenelemente a und b (erster und zweiter Parameter) und dem Zeiger `client` (dritter Parameter) aufgerufen und sollte einen Wert größer als 0 zurückliefern, wenn der Wert im Datenbereich von a größer ist als der im Datenbereich von b, einen Wert kleiner als 0, wenn der Wert in a kleiner ist als der in b und den Wert 0, wenn die Werte in a und b gleich groß sind.
Ergebnis	Die Funktion hat kein Ergebnis.

dcl_merge

Name	"merge two doubly connected lists" — Vereinigen zweier doppelt verketteter Listen
Definition	`int dcl_merge (DCLIST *dst, DCLIST *src);`
Beschreibung	Die beiden doppelt verketteten Listen `src` (source) und `dst` (destination) werden vereinigt, d.h. alle Listenelemente der Liste `src` werden in die Liste `dst` übertragen, und zwar werden sie hinter dem letzten Element der Liste `dst` angehängt. Folglich befinden sich nach der Vereinigung der beiden Listen alle Elemente aus `dst` vor allen Elementen aus `src`. Die nach dieser Übertragung leere Liste `src` wird gelöscht. Falls der Cursor der Liste `dst` auf ein Element der Liste zeigte (sich also nicht hinter dem letzten Element befand), wird er nicht verändert, zeigt also auf das gleiche Element wie vor der Vereinigung der beiden Listen. Stand er dagegen hinter dem letzten Element, so zeigt er nach der Vereinigung auf das erste aus der Liste `src` übertragene Element. Gelingt das Vereinigen nicht, bleiben die Listen unverändert.
Bemerkung	Das Vereinigen der Listen kann nur bei Verwendung der Vektorimplementierung (Datei `dcl_vec.c`) fehlschlagen.
Ergebnis	Falls das Vereinigen gelang, wird 0, anderenfalls ein Wert ungleich 0 zurückgegeben.

5 Implementierung als echte Liste

```
typedef struct dcle {           /* --- list element --- */
  struct dcle *succ;            /* pointer to successor */
  struct dcle *pred;            /* pointer to predecessor */
} DCLE;                         /* (list element) */

struct dclist {                 /* --- doubly connected list --- */
  int         len;              /* length (number of elements) */
  DCLE        *frst;            /* pointer to first   element */
  DCLE        *last;            /* pointer to last    element */
  DCLE        *curr;            /* pointer to current element */
  DCLE_FN     *delfn;           /* deletion function */
};                              /* (doubly connected list) */
```

6 Implementierung als Vektor

```
struct dclist {                 /* --- doubly connected list --- */
  DCLE_FN *delfn;               /* deletion function */
  int     curr;                 /* index of current element */
  int     vsz;                  /* size of vector */
  int     len;                  /* length (number of elements) */
  void    **vec;                /* vector of list elements */
};                              /* (doubly connected list) */
```