

Doppelt verkettete Liste

Doppelt verkettete Listen verhalten sich im Grunde wie einfach verkettete Listen mit einem zusätzlichen Pointer auf das vorherige Element. Das bringt einige Vorteile. Zum Beispiel ist es in solch einer Datenstruktur möglich ein bestimmtes Element der Liste zu entfernen, ohne vorher durch die Liste zu iterieren, um den Vorgänger zu finden, da man auch rückwärts gehen kann. Allerdings hat man auch einen höheren Speicherbedarf, aufgrund der zusätzlichen Pointer. Diese verkomplizieren auch das Einfügen und Löschen von Elementen. Um die Datenstruktur zu erhalten, muss man daher besonders darauf achten, keine Dangling-Pointer zu hinterlassen.

Im Folgenden wollen wir uns einmal konzeptionell ein Modul für die Arbeit mit doppelt verketteten Listen anschauen. Dafür wollen beispielhaft einige Subroutinen und Funktionen implementieren:

1. Einen Datentypen *elem* mit einer *content* Komponente und zwei *Pointern* vom Typ *elem* auf das nächste und vorherige Element der Liste.
2. Einen Datentypen *list* mit einer *Integer* Komponente für die Länge der Liste und einem *head* und *tail* Pointer vom Typ *elem*
3. Eine *Subroutine* *INIT*, um eine leere Liste zu initialisieren.
4. Eine logische *Funktion* *EMPTY*, die zurückgibt, ob eine übergebene Liste leer ist.
5. Die *Subroutinen* *PUSH* und *POP*, die jeweils am Anfang der Liste ein Element einfügen oder löschen können.
6. Eine *Funktion* *HEAD*, die den Inhalt/Wert des *head* Elements zurückgibt.
7. Die *Subroutinen* *INJECT* und *EJECT*, die jeweils am Ende der Liste ein Element einfügen oder löschen können.
8. Die *Subroutine* *INS_AFTER*, die in der Liste nach einem bestimmten Element ein neues einfügt.
9. Die *Subroutine* *DEL_ELEM*, die in der Liste ein bestimmtes Element löschen kann.