

Einfach verkettete Listen

Freitag, 19. Januar 2024 16:36

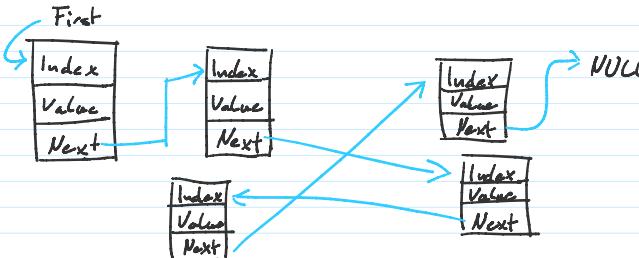
Einführung:

- dynamische Datenstruktur
 - das heißt, die Größe der Daten wächst
- Voraussetzung: dynamische Speicherverwaltung

Datenlemente mit Zeigern verketten

Beispiel:

```
struct ListElement{
    //Daten
    int Index;
    double Value;
    struct ListElement *Next;
};
```



einfach verkettete Liste

```
#ifndef EINFACHVERKETTETELEISTE_H
#define EINFACHVERKETTETELEISTE_H

typedef struct SLE{
    int Index;
    double Value;
    struct SLE *Next;
}ListElement;

extern ListElement *First, *Last;
```

//Deklaration verschiedener Funktionen

```
ListElement *createListElement(int, double);
void appendListElement(ListElement *);
void insertListElement(ListElement *, int (*)(ListElement *, ListElement *));
ListElement *removeListElement(ListElement *, int (*)(ListElement *, ListElement *));
ListElement *getFirstElement();
```

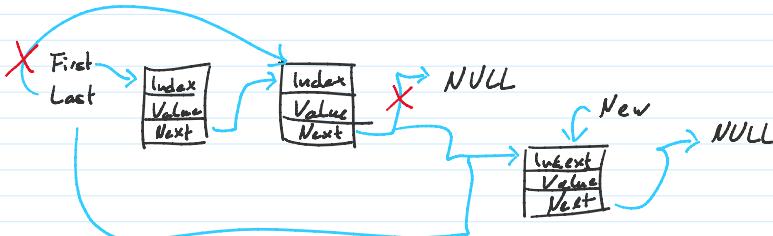
Neues Listenelement an Liste anhängen

- Fallunterscheidung: „Liste ist leer“ oder „Liste ist nicht leer“
- Fall 1: Liste ist leer:
 - Abfrage: if(First == NULL)
 - First und Last müssen auf neues Listenelement zeigen



- Fall 2: Liste ist nicht leer:

- Next-Zeiger des letzten Listenelements (Last → Next) muss auf neues Listenelement zeigen
- Last muss auf neues Listenelement zeigen



Neues Listenelement in Liste einfügen

Fallunterscheidung

- Fall 1: Liste ist leer

- identisch mit Fall 1 von „neues Listenelement an Liste anhängen“
First = Last = New;

- Fall 2: vor dem ersten Element einfügen:

- Abfrage ob vor dem ersten Element eingefügt werden muss
if(compare(First, New) >= 0)
- Next-Zeiger vom neuen Element auf bisher erstes Element zeigen
- First-Zeiger auf neues Element zeigen

einfachverketteteliste.c

```
#include <stdio.h>
#include <stdlib.h>
#include "einfachverketteteleiste.h"

static ListElement *First = NULL, *Last = NULL;

ListElement *createListElement(int i, double v){
    ListElement *New = malloc(sizeof(ListElement));
    if(New){
        New->Index = i;
        New->Value = v;
        New->Next = NULL;
    }
    return New;
}

void appendListElement(ListElement *new){
    if(First){
        if(First == NULL)
            First = Last = New;
        else
            Last = Last->Next = New;
    }
}

void insertListElement(ListElement *New, int (*cmpfct)(ListElement *, ListElement *)){
    ListElement *tmp = First;
    if(New){
        if(First == NULL)
            First = Last = New;
        else if(compare(First, New) >= 0){
            New->Next = First;
            First = New;
        }
        else if(compare(Last, New) <= 0)
            Last = Last->Next = New;
        else{
            while(tmp->Next){
                if(compare(tmp->Next, New) > 0){
                    New->Next = tmp->Next;
                    tmp->Next = New;
                    break;
                }
                tmp = tmp->Next;
            }
        }
    }
}

ListElement *removeListElement(ListElement *Remove, int (*compare)(ListElement *, ListElement *)){
    if(Remove){
        if(First == NULL)
            ;
        else if(compare(First, Remove) == 0){
            tmp = First;
            First = First->Next;
            if(First == NULL)
                Last = NULL;
        }
        else{
            tmp = prv->Next;
            while(tmp){
                if(compare(tmp, Remove) == 0){
                    prv->Next = tmp->Next;
                    if(prv->Next == NULL)
                        Last = prv;
                    break;
                }
                prv = prv->Next;
                tmp = tmp->Next;
            }
        }
    }
    return tmp;
}

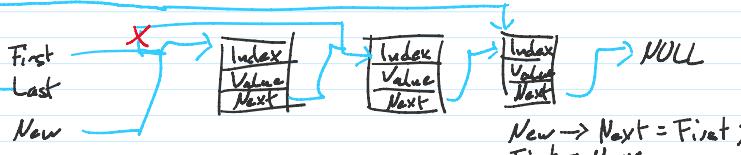
ListElement *getFirstElement(){
    return First;
}
```

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "einfachverketteteleiste.h"
#include "tools.h"

int compareIndex(ListElement *LE1, ListElement *LE2){
    return (LE1->Index - LE2->Index);
```

- if (compare(First, New) ≥ 0)
 - Next - Zeiger vom neuen Element auf bisher erstes Element zeigen
 - First - Zeiger auf neues Element zeigen

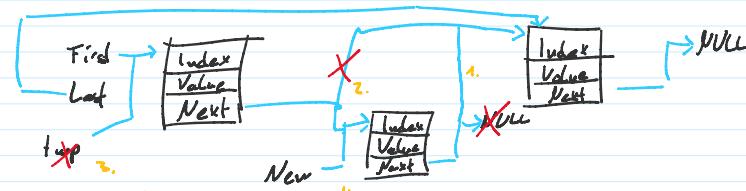


- Fall 3: nach dem letzten Element anfügen

- Abfrage ob nach dem letzten Element angefügt werden muss:
if (compare(Last, New) ≤ 0)
 - Next - Zeiger vom bisher letzten Element auf neues Element zeigen
 - Last - Zeiger auf neues Element zeigen
 - \hookrightarrow identisch mit Fall 2 von „Neues Listenelement an Liste anfügen“
 $\text{Last} = \text{Last} \rightarrow \text{Next} = \text{New};$

- Fall 4: unten hin einfügen

- Abfrage ob vor dem nächsten Element eingefügt werden muss:
if (compare(Temp->Next, New) > 0)
 - Next - Zeiger vom neuen Element auf nächstes Element zeigen
 - Next - Zeiger vom aktuellen Element auf neues Element zeigen



Listenelement aus Liste entfernen

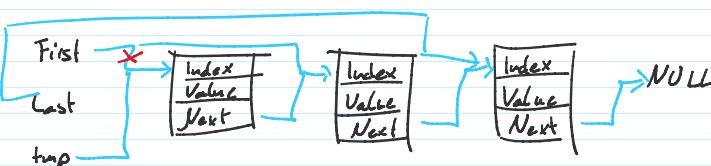
Fallunterscheidung:

- Fall 1: Liste ist leer

- wir passiert
return NULL

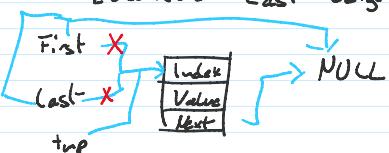
- Fall 2: erstes Element entfernen (Sonderfall: einziges Element entfernen)

- Abfrage ob erstes Element entfernt werden muss
if (compare(First, Remove) ≤ 0)
 - temp - Zeiger auf erstes Element zeigen
 - First - Zeiger auf First \rightarrow Next zeigen



temp = First;
First = First \rightarrow Next;
return temp;

- Sonderfall: erstes Element == Letztes Element
 \hookrightarrow zusätzlich Last - Zeiger auf NULL zeigen



temp = First;
First = First \rightarrow Next;
if (First == NULL)

```
#include <csio.h>
#include <stdlib.h>
#include "einfachverketteteliste.h"
#include "tools.h"

int compareIndex(ListElement *LE1, ListElement *LE2){
    return (LE1->Index - LE2->Index);
}

int main(){
    ListElement *tmp = NULL;

    insertListElement(createListElement(5, 17.3), compareIndex);
    insertListElement(createListElement(2, 24.7), compareIndex);
    insertListElement(createListElement(4, 3.9), compareIndex);
    insertListElement(createListElement(1, 34.4), compareIndex);
    insertListElement(createListElement(3, 53.1), compareIndex);
    insertListElement(createListElement(6, 49.6), compareIndex);

    tmp = getFirstElement();
    title("Liste der Daten", "="); // Eigene Funktion aus tools.h
    while(tmp){
        printf("%3i | %5.1f\n", tmp->Index, tmp->Value);
        tmp = tmp->Next;
    }

    while(tmp = getFirstElement())
        free(removeListElement(tmp, compareIndex));
}

return 0;
}
```

main.c , zweites Beispiel

```
#include <stdio.h>
#include <stdlib.h>
#include "einfachverketteteliste.h"

int compareIndex(ListElement *LE1, ListElement *LE2){
    return (LE1->Index - LE2->Index);
}

int compareValue(ListElement *LE1, ListElement *LE2){
    return (LE1->Value - LE2->Value);
}

int main(){
    ListElement *tmp = NULL;
    ListElement Loeschen;
    int i;

    insertListElement(createListElement(5, 17.3), compareValue);
    insertListElement(createListElement(2, 24.7), compareValue);
    insertListElement(createListElement(4, 3.9), compareValue);
    insertListElement(createListElement(1, 34.4), compareValue);
    insertListElement(createListElement(3, 53.1), compareValue);
    insertListElement(createListElement(6, 49.6), compareValue);

    tmp = getFirstElement();
    title("Liste der Daten", "=");
    while(tmp){
        printf("%3i | %5.1f\n", tmp->Index, tmp->Value);
        tmp = tmp->Next;
    }

    for(i = 1; i <= 6; i++){
        Loeschen.Index = i;
        free(removeListElement(&Loeschen, compareIndex));
    }

    return 0;
}
```

```

tmp = First;
First = First->Next;
if (First == NULL)
    Last = NULL;
return;

```

• Fall 3: sonstige Entfernen:

- zwei temporäre Zeiger nötig: tmp (akt.) und prv (vorheriges Element)
- prv zeigt am Anfang auf First; tmp auf das nächste (aktuelle) Element
- Abfrage, ob aktuelles Element entfernt werden muss
if (compara (tmp, Remova) == 0)
- mit prv->Next auf Nachfolger vom aktuellen Element (tmp->Next) zeigen
- Spezialfall: letztes Element wird entfernt
if (prv->Next == NULL)
- zusätzlich Last auf prv zeigen

