Einfach verkettete Listen

Datenstruktur definieren

```
typedef struct sData
{
  int Index;
  double Value;
  struct sData *Next;
};
```

By **TimSch**
cheatography.com/timsch/

Published 18th March, 2018.
Last updated 18th March, 2018.

Einfach verkettete Listen (cont)

Globale Zeiger auf Listenanfang und Listenende

```
extern TData *First = NULL;
extern TData *Last = NULL;
```

Schlüsselwort extern, damit die Zeiger über alle Quelltextdateien verfügbar sind. Alternativ in main definieren.

Einf

F

By **TimSch**
cheatography.com/timsch/

Published 18th March, 2018.
Last updated 18th March, 2018.

Einfach verkettete Listen (cont)

Neues Element anhängen

```
int appendInList(TData *Neu)
{
  //prüfen, ob neues Element
existiert
  if (Neu == NULL)
    return 0;

  //Neues Element ist neues
Listenende
  Neu->Next = NULL;

  if (First == NULL)
    //Fall 1: Liste ist leer
    First = Last = Neu;
  else
    //Fall 2: Liste ist
nicht leer
    Last = Last->Next = Neu;
  return 1;
}
```

Einfach verkettete Listen (cont)

Neues Element (sortiert) einfügen

```
int insertInList(TData *Neu)
{
  TData *tmp = First;
  //prüfen, ob neues Element existiert
  if(Neu == NULL)
    return 0;
  if(First == NULL)
  { //Fall 1: Liste ist leer
    Neu->Next = NULL;
    First = Last = Neu;
    return 1;
  }
  if (First->Value >= Neu->Value(
  { //Fall 2: am Listenanfang einfügen
    Neu->Next = First;
    First = Neu;
    return 1;
  }
  if (Last->Value <= Neu->Value)
  { //Fall 3: am Listenende anhängen
    Neu->Next = NULL;
    Last = Last->Next = Neu;
    return 1;
  }
  //Fall 4: zwischen zwei Elemente einfüge
  while (tmp->Next != NULL)
  { //prüfen, ob neues Listenelement vor d
element eingefügt werden muss
    if (tmp->Next->Value >= Neu->Value)
    {
      Neu->Next = tmp->Next;
      tmp->Next = Neu;
      return 1;
    }
    return 0;
  }
```

Einfach verkettete Listen (cont)

Element löschen

```
TData *removeFromList(int delIndex)
{
  TData tmp = NULL, prev = NULL;
  // Fall 1: Liste leer?
  if (First == NULL)
    return NULL;

  // Fall 2: Listenanfang entfernen?
  if (First->Index == delIndex)
  {
    tmp = First;
    // nur ein Element in Liste?
    if (Last == First)
      Last = NULL;
    First = First->Next;
    return tmp;
  }

  // Fall 3: zu entfernendes Element suchen
  prev = First;
  tmp = prev->Next;
  while (tmp != NULL)
  {
    if (tmp->Index == delIndex)
    {
      prev->Next = tmp->Next;
      if (tmp == Last) // letztes Element entfernt?
        Last = prev;
      return tmp;
    }
    prev = tmp;
    tmp = tmp->Next;
  }
  return NULL;
}
```

Doppelt verkettete Listen

An Liste anhängen

```
int appendInList(T
{
  //prüfen, ob neu
  if (Neu == NULL)
    return 0;

  if (First == NUL
    //Fall 1: List
    Neu->Next = Ne
    First = Last =
  else
    //Fall 2: List
    Neu->Next = NU
    Neu->Prev = La
    Last = Last->N
  return 1;
}
```

Doppelt verkettete Listen (cont)

In Liste (sortiert) einfügen

```
int insertInList(TData *Neu)
{
 TData *tmp = First;

 //prüfen, ob neues Element existiert
 if(Neu == NULL)
   return 0;

 if(First == NULL)
 { //Fall 1: Liste ist leer
   Neu->Next = Neu->Prev = NULL;
   First = Last = Neu;
   return 1;
 }

 if (First->Value >= Neu->Value(
 { //Fall 2: am Listenanfang einfügen
   Neu->Next = First;
   Neu->Prev = NULL;
   First = First->Prev = Neu;
   return 1;
 }

 if (Last->Value <= Neu->Value)
 { //Fall 3: am Listenende anhängen
   Neu->Prev = Last;
   Neu->Next = NULL;
   Last = Last->Next = Neu;
   return 1;
 }

 //Fall 4: zwischen zwei Elemente einfügen
 while (tmp->Next != NULL)
 { //prüfen, ob neues Listenelement vor dem nächsten Listenelement
eingefügt werden muss
    if (tmp->Next->Value >= Neu->Value)
    {
      Neu->Next = tmp->Next;
      Neu->Prev = temp;
      tmp->Next = temp->Next->Prev = Neu;
      return 1;
    }
    tmp = temp->Next;
 }
 return 0;
}
```

Doppelt verkettete Listen (

Aus Liste entfernen

```
TData *re
delIndex)
{
 TData *t
NULL;
 // Fall
 if (Firs
   return

 // Fall
entferner
 if (Firs
 {
   tmp =
   First
   // nur
   if (F
     Last
   else
     Firs
   return
 }

 // Fall
entferner
 if (Last
 {
   tmp =
   Last =
   Last->
   return
 }

 //Fall 4
Element s
 tmp = F
 while (t
 {
   if (tm
   {
     prev
     prev
     prev
     retu
   }
   tmp =
 }
 return N
}
```

Doppelt verkettete Listen (cont)

Alle Elemente auflisten

```
typedef enum {forward, backward} TDirection;

void printList(TDirection Direction)
{
  TData *tmp = (Direction == forward) ? First : Last;
  int AnzahlElemente = 0;

  printf("\nIndex ");
  while (tmp)
  {
    printf("| %3i ", tmp->Index);
    tmp = (Direction == forward) ? tmp->Next : tmp->Prev;
    AnzahlElemente++;
  }

  printf("\n------");
  while (AnzahlElemente--)
    printf("------");

  printf("\nWert ");
  tmp = (Direction == forward) ? First : Last;
  while (tmp != NULL)
  {
    printf("| %3.0f ", tmp->Value);
    tmp = (Direction == forward) ? tmp->Next : tmp->Prev;
  }
  printf("\n");
}
```