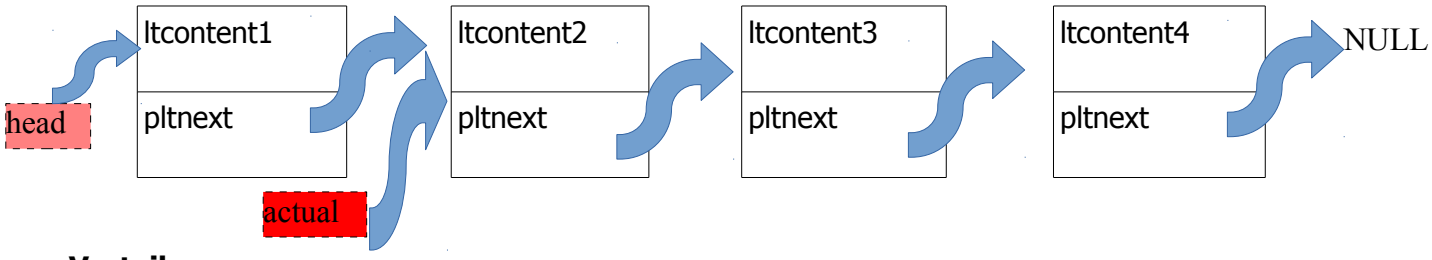
# Die lineare Liste

Ein **Listenelement** besteht aus dem Inhalt und dem Verweis auf das nächste Element.

| ltcontent |
| --- |
| pltnext |

Die einzelnen Elemente liegen **nicht** wie bei Feldern direkt im Speicher hintereinander sondern an beliebigen Stellen und werden durch den Verweis des Vorgängers (vorgaenger->pnaechstes speichert die Speicheradresse des Elementes) auffindbar.

Die **Liste** besteht letztendlich nur
aus einem Zeiger auf das erste Element – diesen nennen wir list->head –
und einem Zeiger mit dem man durch die Elemente wandert – diesen nennen wir list->actual.




**Vorteile:**
Operationen wie Einfügen oder Löschen werden einfacher realisierbar. Es werden gegenüber Feldern Verschiebeoperationen gespart. Die Lineare Liste ist zudem nicht von vornherein auf eine bestimmte Größe festgelegt sondern kann dynamisch beliebig erweitert werden. Nur die vorhandene Speichergröße des Computers setzt hier Grenzen.

Die lineare Liste ist ein abstrakter Datentyp und hat folgende Eigenschaften:

**Eigenschaften eines ADT (Abstrakter Datentyp)**

Die Eigenschaften eines gut programmierten ADT und in den meisten Fällen auch einer gut definierten Datenstruktur sind:

- ***Universalität*:**
  Der ADT wird einmal entworfen und implementiert und ist einsetzbar in jedem beliebigen Programm.
- ***Präzise Beschreibung*:**
  Eine eindeutig und vollständig beschriebene Schnittstelle zwischen Implementierung und Anwendung.
- ***Einfachheit*:**
  Der ADT übernimmt selbst die Verwaltung des für ihn relevanten Speichers.
- ***Kapselung*:**
  Da der Zugriff nur über die festgelegten Operationen (Methoden) erfolgt, sind die Daten nach außen gekapselt. Die innere Realisierung bleibt verborgen. Der Anwender soll sehr genau wissen, was ein ADT tut, aber keinesfalls, wie er es tut.
- ***Geschütztheit*:**
  Der Eingriff auf die innere Struktur durch den Anwender soll verhindert werden.
- ***Modularität*:**
  Das Programmieren wird übersichtlicher und Programmteile sind leichter austauschbar.

**Die Lineare Liste in C:**
**DATEN:**

```
// Vom Benutzer angelegter Elementtyp (Inhalt) muss immer ans Projekt angepasst werden.
typedef struct
{
   ........
}l_elementtype;

// rekursiv definierte Struktur
// Listenobjekte bestehend aus einem Elementtyp und einem Verweis auf ein weiteres Listenobjekt
struct l_basisobjecttype
{
        l_elementtype ltcontent ;                  //Inhalt
        struct l_basisobjecttype *pltnext ;        // Pointer auf nächstes Listenelement
};

//Um sich das struct zu sparen wird der Pointer auf die Struktur als eigener Datentyp definiert
typedef struct l_basisobjecttype *l_objecttype;

// Die eigentliche Liste besteht aus einem Zeiger auf den Listenanfang: head
// und einem Zeiger auf das gerade betrachtete Listenelement: actual
typedef  struct
{
         l_objecttype head , actual;
}l_listtype;
```

**METHODEN:**

```
//Am Anfang wird eine leere Liste angelegt:
    l_listtype l_create();   // erzeugt eine neue leere Liste (list→head und list->actual zeigen auf NULL)

//Bevor ein Element in die Liste eingefügt werden kann, muss es existieren, d.h. der Speicherplatz
//für so ein Element muss bereitgestellt werden:
    l_objecttype l_create_newelement();   //erzeugt neues Listenelement und liefert den Pointer zurück

//Den Inhalt des aktuellen Elementes mit Werten belegen(muss immer angepasst werden):
void l_setcontent(l_elementtype *newelement);

//Den Inhalt des aktuellen Elementes ansehen:
    l_elementtype * l_getcontent (l_listtype lList) ;  // Pointer auf den Inhalt wird zurück gegeben

//Das aktuelle Element aus der Liste löschen:
   void l_delete (l_listtype *lList ) ;  // das neue aktuelle Element ist das erste Element

//Den Inhalt des aktuellen Elementes updaten:
    void l_update (l_listtype *lList , l_elementtype newelement ) ;  //newelement ersetzt den alten Inhalt

//Das aktuelle Element auf den Listenanfang setzen:
    void l_reset (l_listtype *lList ) ;

//Das aktuelle Element eins weiter setzen:
    void l_next (l_listtype *lList ) ;    // list->actual auf den Nachfolger setzen

//Ein neues Element ans Ende der Liste hängen:
    void l_last (l_listtype *lList , l_objecttype newelement) ;   //

//Ein neues Element vor dem aktuellen Element in die Liste einfügen:
    void l_insert (l_listtype *lList , l_objecttype newelement ) ;   //
```

**//Überprüfen, ob das Ende der Liste erreicht wurde:**
int l_end (l_listtype lList ) ; // gibt eine 1 zurück, falls das aktuelle Element das letzte ist

**//Überprüfen, ob die Liste leer ist:**
int l_empty (l_listtype lList ) ; // gibt eine 1 zurück, falls die Liste leer ist

**Aufgaben**

1) **Die zur Verfügung gestellte Datei linliste.c realisiert noch nicht alle angedachten Funktionen (siehe linliste.h). Schauen Sie sich die realisierten Funktionen an.**
2) **Erläutern Sie die Funktionen mit Hilfe der Karten an der Tafel.**
3) **Ergänzen Sie die noch fehlenden Funktionen und beschreiben Sie die Vorgehensweise.**
4) 

**a)**

**Projekt: Telefonliste**

Eine nach dem Nachnamen alphabetisch sortierte Telefonliste der Klasse mit Name, Vorname, Festnetz und Mobilnummer soll von einem Programm abgefragt werden. Anschließend soll Sie auf dem Monitor ausgegeben und als Textdatei abgespeichert werden.

1. Entwickeln Sie zu diesem Problem den passenden Typ **l_elementtype**.
2. Passen Sie die Funktion „void l_setcontent(l_elementtype *newelement);" an.
3. Schreiben Sie ein Programm, welches die geforderten Daten vom Benutzer einliest und zum Schluss die sortierte Liste auf dem Monitor ausgibt.
4. Entwickeln und realisieren Sie eine Funktion, die die Liste in eine Textdatei schreibt.
5. Entwickeln und realisieren Sie eine Funktion, die die Liste aus einer Textdatei ausliest.

**Vereinbarung:** Abgabe eines ausführlichen Laborberichtes über das Projekt mit einem Kapitel der theoretischen Erläuterung über das Auffinden der richtigen Stelle zur Einsortierung des Namens und die Darstellung der Überlegungen zum Abspeichern in einer Datei.

**b)**

**Projekt: Polynome**

Polynome sind (wie aus dem Mathematikunterricht bekannt) ganzrationale Funktionen der Form:

$$f(x)=\sum_{i=0}^{n} a_i x^i = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_2 x^2 + a_1 x + a_0 ; a_i \in \mathbb{R}, i \in \mathbb{N}$$

$$Beispiel: f(x)=3x^4+(-2)x^3+9x^0$$

Polynome lassen sich gut durch lineare Listen speichern. Man fügt nur die Glieder der Polynome als Elemente der Liste zu, die auch wirklich besetzt sind.

1. Entwickeln Sie zu diesem Problem den passenden Typ **l_elementtype**.
2. Passen Sie die Funktion „void l_setcontent(l_elementtype *newelement);" an.
3. Schreiben Sie ein Programm, welches ein Polynom vom Benutzer einliest und auf dem Monitor ausgibt.
4. Entwickeln und realisieren Sie eine Funktion, die ein Polynom ableitet(differenziert).
5. Entwickeln und realisieren Sie eine Funktion, die zwei Polynome addiert/subtrahiert.

**Vereinbarung:** Abgabe eines ausführlichen Laborberichtes über das Projekt mit einem Kapitel der theoretischen Erläuterung über Polynome und die Berechnung der Ableitung so wie die Darstellung der Überlegungen zur Addition zweier Polynome – welche Bedingungen überprüft werden müssen.