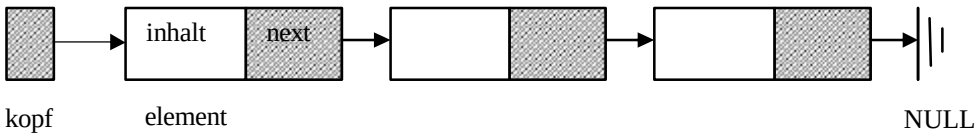
Einfach verkettete Listen

1. Veranschaulichung



Eine (lineare) Liste ist eine Datenstruktur, bei der jedes Element (außer dem letzten) genau einen Nachfolger hat. Das erste Element bezeichnen wir mit *kopf*. Jedes Listenelement (außer dem *kopf*) besteht aus (mindestens) zwei Komponenten, der Inhaltskomponente und der Zeigerkomponente (Verweis auf das folgende Element).

2. Vereinbarung und Zugriff

```
//Definition des Strukturtyps tElement
struct  tElement
{       char inhalt[20];
        tElement* next;
};

//Vereinbarung einer Zeigervariablen von obigem Typ
tElement *satz;
```

Auf die Komponenten dynamisch erzeugter Strukturen wird über den **Pfeil-Operator** -> zugegriffen, z. B.:

```
cin >> neu -> inhalt;
neu -> next = NULL;
```

3. Funktion zum Aufbau einer Liste (Einfügen am Ende)

```
void aufbau(tElement *&kopf)
{
        tElement *neu, *ende;
        cout << "Satz Wort fuer Wort eingeben" << endl;
        cout << "Satz mit einzelnem Punkt abschliessen" << endl;
        kopf = NULL;                    //kopf ist zunaechst leerer Zeiger
        ende = NULL;
        do
        {
                neu = new tElement;     //neuen Zeiger initialisieren
                cin >> neu->inhalt;     //Wertzuweisungen fuer die
                neu -> next = NULL;     //Zeigerkomponenten inhalt u. next
                if (kopf == NULL) kopf = neu;
                else ende->next = neu;  //neues El. ans Listenende 'koppeln'
                ende = neu;             //neues Ende festlegen
        }
        while (neu->inhalt[0]!='.');    //Abbruchbedingung
}
```

4. Funktion zur Ausgabe der Liste

```
void ausgabe(tElement *kopf)
{
        tElement *aktuell;
        aktuell = kopf;
        while (aktuell != NULL)
        {
                cout << aktuell->inhalt << " ";
                aktuell = aktuell->next;        //Zeiger 'weiterschieben'
        }
}
```

Aufgabe 1:
Schreiben Sie ein Programm, das obige Vereinbarungen und Funktionen nutzt, um eine einfach verkettete Liste aufzubauen und auf dem Bildschirm auszugeben. Die Elemente der Liste sollen Zeichenketten sein, z. B. ‚Das' ‚ist' ‚ein' ‚Satz' ‚.' (als Abbruchbedingung für das Satzende fordere man die Eingabe eines einzelnen Punktes).

Aufgabe 2:
Schreiben Sie eine **rekursive** Funktion ***ausgaberek*** zur Ausgabe einer Liste und arbeiten Sie diese in das Programm aus Aufgabe 1 ein. Setzen Sie dabei folgende Definition um: Ist die Liste nicht leer (***kopf != NULL***), dann gib den Inhalt des Listenkopfes auf dem Bildschirm aus und ruf die Funktion für die Restliste auf (***ausgaberek(kopf->next)***).

5. Funktion zum Einfügen von neuen Elementen in eine Liste

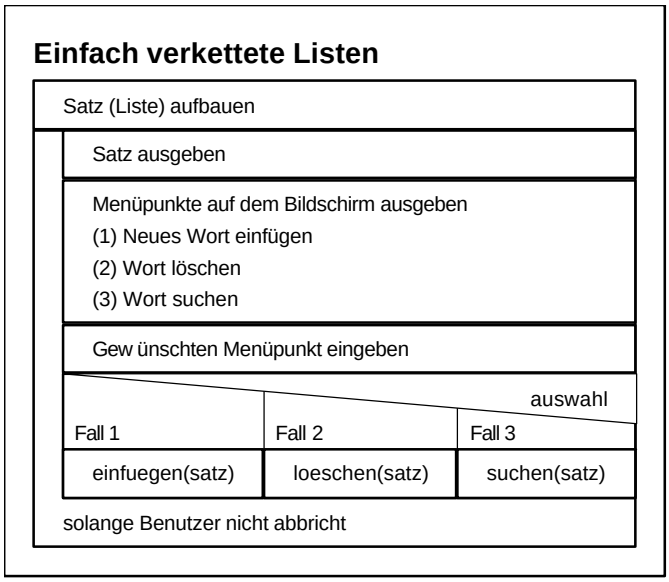
Folgende Funktion dient dem Einfügen eines neuen Listenelementes an einer gewünschten Stelle.

```
void einfuegen(tElement *& kopf)
{
        tElement *aktuell, *hilf;
        char nachwort[20];
        cout << endl << "Nach welchem Wort soll ein neues eingefuegt werden? ";
        cin >> nachwort;
        aktuell = kopf;
        while (aktuell != NULL)
        {
        //Vergleich von 2 Strings nur mit strcmp aus <string.h> moeglich
        if (strcmp(aktuell->inhalt, nachwort) == 0)
                {
                hilf = new tElement;
                cout << endl << "Welches Wort soll eingefuegt werden? ";
                cin >> hilf->inhalt;
                hilf->next = aktuell->next;
                aktuell->next = hilf;
                }
        aktuell = aktuell->next;
        }
}
```

Aufgabe 3:
Ergänzen Sie das Programm aus Aufgabe 1 durch obige Funktion und rufen Sie diese Funktion im Programm an geeigneter Stelle auf.

Aufgabe 4:
Erstellen Sie Funktionen zum Löschen und Suchen von Listenelementen und erweitern Sie Ihr Programm dementsprechend.

Aufgabe 5:
Setzen Sie folgendes Struktogramm in Ihrem Programm um:



Aufgabe 6:
a) Was bewirkt folgende Funktion?

```
void wasmacheich(int inZahl, tElement *&kopf)
{       tElement *aktuell, *neu; bool gefunden;
        neu = new tElement;
        neu->inhalt = inZahl;
        if (kopf == NULL)
        {       neu->next = kopf; kopf = neu;}
        else    if (kopf->inhalt > inZahl)
                {       neu->next = kopf;  kopf = neu;}
                else
                {       gefunden = false; aktuell = kopf;
                        while (aktuell->next != NULL && !gefunden)
                        {       if (aktuell->next->inhalt > inZahl) gefunden = true;
                                else aktuell = aktuell->next;
                        }
                        if (gefunden)
                        {       neu->next = aktuell->next; aktuell->next = neu;}
                        else
                        {       aktuell->next = neu;  neu->next = NULL;}
                }
}
```

b) Schreiben Sie ein Programm, das die Struktur *tElement* wie folgt definiert und obige Funktion nutzt: ***struct tElement {int inhalt; tElement *next;};*** Kommentieren Sie reichlich!