```c
#include <stdio.h>
#include <string.h>
#include <malloc.h>

typedef struct item_type {     /* Datenstruktur */
       char *c;                /* Datenmember   */
       /* ...    weitere Datenmember moeglich   */
} item_type;

typedef struct list {       /* Listenelement          */
       item_type item;      /* struct der Datenmember */
       struct list *next;   /* point to successor     */
} list;

/* Vergleich i1->c == i2->c , wenn gleich, dann 1, sonst 0 */
int equal(item_type *i1, item_type *i2){
       if(i1==NULL && i2==NULL) return 1;       /* beide Zeiger 0 */
       if(i1==NULL || i2==NULL) return 0;       /* nur 1 Zeiger 0 */
       if(i1->c==NULL && i2->c==NULL) return 1; /* i1->c und i2->c sind 0     */
       if(i1->c==NULL || i2->c==NULL) return 0; /* nur i1->c oder i2->c ist 0 */
       if(!strcmp(i1->c, i2->c)) return 1;      /* i1->c == i2->c ? */
       return 0;                                /* i1->c != i2->c   */
}

/* Suche in Liste l das item_type x, wenn gefunden: Rueckgabe Adresse x , sonst 0 */
list *search_list(list *l, item_type x) {
       if (l == NULL) return(NULL);  /* Abbruch, Rueckgabe 0, wenn x nicht in Liste l */
       if (equal(&(l->item), &x))    /* Adressen von l->item und x identisch ? */
              return l;              /* Rueckgabe l  (== &x), d.h. Adresse item */

       return search_list(l->next, x); /* rekursive Suche in Restliste l->next */
}

/* Einfuegen von item_type x an den Anfang der Liste *l */
void insert_list(list **l, item_type x) { /* Parameter l ist Adr. der Adr. auf list */
       list *p;                      /* lokale, temporaere Zeigervariable p auf list */
       p = (list *)malloc( sizeof(list) );   /* p zeigt auf neues Listenelement */
/*
wenn x.c == 0, dann p->item.c = 0 , sonst Anlegen von p->item.c im Laenge von x.c + 1
(+1 wegen '\0') und kopieren von x.c mit strcpy nach Adresse p->item.c

Folgende Anweisung kann man auch als Folge von einfachen Einzelanweisungen schreiben
*/
       p->item.c = x.c ? strcpy((char *)calloc(strlen(x.c)+1, sizeof(char)), x.c) : 0;

/* alternativ zur vorherigen Anweisung waere folgende Anweisung p->item = x; unsicher */
/* p->item = x; */ /* Falls x.c dynamisch mit calloc oder malloc angelegt, dann zeigen
   x.c und (p->item).c  auf den gleichen dynamischen Speicher, das wird vom
   Programmierer schnell "vergessen". Im Falle der Freigabe oder Wertaenderung von x
   oder p->item "weiss" die andere Variable davon nichts, was zur fehlerhaften
   Weiterverwendung des nicht mehr existierenden oder veraenderten Speicherplatzes
   fuehren kann, haeufig ist ein Speicherzugriffsfehler mit Programmabbruch die Folge
*/
```

```
        p->next = *l; /* Nachfolger des neuen ersten Listenelements wird Adresse des
                         bisherigen erstes Elementes *l */
        *l = p;       /* Listenzeiger *l zeigt auf neues erstes Element p */
}

/* einmaliges Entfernen eines Listenelementes mit Wert x aus Liste *l */
void delete_list(list **l, item_type x) { /* Parameter l ist Adr. der Adr. auf list */
        list *p;                      /* lokale, temporaere Zeigervariable p auf list */
        list *last = NULL;            /* weitere Zeigervariable, predecessor pointer */
        if(!(*l) || !(&x)) return;    /* leere Liste oder x ist deref. Nullzeiger */
        p = *l;                       /* p zeigt auf erstes Listenelement */
        /* find item to delete */
        while (p && !equal(&(p->item), &x)) { /* wenn p ex. und equal 1 liefert, dann
                                                 Abbruch der Schleife */
                last = p;                    /* last = aktuelles p */
                p = p->next;                 /* p ist Nachfolger von last */
        }

        if(!p && !last) return; /* kann im Algorithmus nicht auftreten, weggelassen */

        if (!p && last) return;              /* x nicht in Liste *l */

        if (p && !last) { *l = p->next;      /* erstes Listenelement p->item == x,
                                                *l zeigt auf p->next */
                          free(p->item.c); free(p);
                          return;
        }

        /* p->item == x fuer nicht erstes Listenelement gefunden */
        if(p && last) { last->next = p->next; /* Verkette Vorgaenger von p (= last)
                                                 mit Nachfolger von p */
                        free(p->item.c); free(p);
                        return;
        }

        return;      /* wegen Compiler-Warnung, wird im Algorithnus nicht erreicht */
}

void show_item(struct list *l){
        if(l){
                printf("data item = %s\n", l->item.c ? l->item.c : "0");
        }
}

void show(struct list *l){
        printf("show list\n");
        while(l){
                printf("data item = %s\n", l->item.c ? l->item.c : "0");
                l=l->next;
        }
        printf("end of list\n");
}
```

```c
void delete_list_all(list **l){
	list *p = *l;
	while(p){
		p = p->next;
		free ((*l)->item.c);  // auch 0 erlaubt
		free(*l);
		*l = p;
	}
	printf("empty list\n");
}

unsigned long long anz(list *l){
	unsigned long long n=0ULL;
	while(l)
		l=l->next, n++;
	return n;
}

int main(){
	list *start = NULL;   /* Liste wird am Anfang durch start repräsentiert */
	item_type p;          /* Datenmember p fuer Liste */
	p.c = "HTW Dresden";
	insert_list(&start, p);
	p.c = "Palucca Hochschule";
	insert_list(&start, p);
	p.c = "TU Dresden";
	insert_list(&start, p);
	p.c = "BA-Dresden";
	insert_list(&start, p);
	p.c=0;
	insert_list(&start, p);
	printf("Anzahl Listenelemente %dULL\n", anz(start));
	show(start);
	p.c = "TU Dresden";
	delete_list(&start, p);
	show(start);
	p.c=0;
	delete_list(&start, p);
	show(start);
	delete_list_all(&start);
	show(start);
	getc(stdin);
	return 0;
}
```

```
/*
Anzahl Listenelemente 5ULL
show list
data item = 0
data item = BA-Dresden
data item = TU Dresden
data item = Palucca Hochschule
data item = HTW Dresden
end of list
show list
data item = 0
data item = BA-Dresden
data item = Palucca Hochschule
data item = HTW Dresden
end of list
show list
data item = BA-Dresden
data item = Palucca Hochschule
data item = HTW Dresden
end of list
empty list
show list
end of list
*/
```