3 Listor

Lista är ett sammanfattande namn för en datastruktur som består *av noll eller flera dataobjekt som är ordnade på något sätt*. För en generell lista ska man kunna sätta in, ta bort eller nå vilket dataobjekt man vill. Man kan använda vektorer för att realisera listor men det är vanligare att man använder *dynamiskt allokerade dataobjekt, som man länkar ihop enligt ovan med pekare till nästa objekt*. Listor kan användas till att avbilda olika typer av samlingar av dataobjekt som köer, stackar mm. Man brukar skilja på listor utgående ifrån hur många pekare man har i posterna. Man pratar om *envägslistor och tvåvägslistor*. I en envägslista kan man bara löpa igenom listan i en riktning, medan man i en tvåvägslista kan flytta sig till länkar både framför och bakom den aktuella.

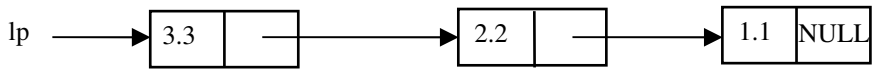
3.1 Envägslistor

I en envägslista har man alltid *en ingång i form av en pekare till listans första länk*. Sedan kan man komma åt de övriga länkarna via nästa-pekare. Beroende på i vilken ordning länkarna stoppas in och tas ut brukar man prata om LIFO-listor och FIFO-listor. I en LIFO (Last In First Out) stoppar man in nya länkar i början och tar ut dem från början. I en FIFO (First In First Out) sätter man in nya länkar på slutet och tar ut dem från början, vilket mera liknar en vanlig normal kö.

3.1.1 LIFO-listor

En lista av LIFO-typ kallas också för en *stack* och används exempelvis av datorer för att spara data under programkörning. Man kan jämföra med en tallrikstapel där man sätter in överst och tar ut överst.

Ex: Skriv ett program som läser in reella tal och stoppar in dessa i en LIFO-lista som skrivs ut vid programslut. Programmet ska innehålla funktioner för att stoppa in tal på stacken (push) och ta bort tal från stacken (pop). Efter inmatning av i ordning 1.1, 2.2 och 3.3 ska listan se ut som:

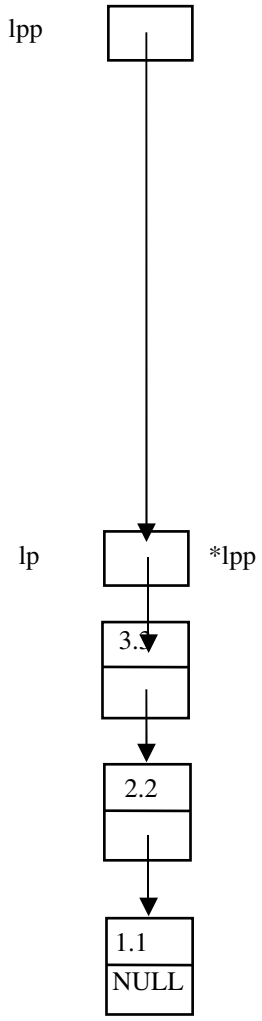



```
#include <stdio.h>
#include <stdlib.h>

struct link
{
   double data;
   struct link *next;
};
```

```
void push(struct link **lpp, double x)
{
   struct link *tp;

   tp = malloc(sizeof(struct link));
   tp->data = x;
   tp->next = *lpp;
   *lpp = tp;
}
```

```
double pop(struct link **lpp)
{
   struct link *tp;
   double x;

   tp = *lpp;
   x = tp->data;
   *lpp = tp->next;
   free(tp);
   return x;
}
```

```
int main()
{
   struct link *lp = NULL;
   double tal;

   printf("Ge tal : ");
   scanf("%lf", &tal);

   while (tal != 0.0)
   {
      push(&lp, tal);
      printf("Ge tal : ");
      scanf("%lf", &tal);

   }

   while (lp != NULL)
      printf("%f\n", pop(&lp));

   return 0;
}
```



OBS! Hur man *skickar adressen för pekaren lp som aktuell parameter och tar emot som en dubbelpekare*. Detta måste man göra eftersom *den aktuella pekarens värde ska förändras*. Pekaren lp får ett nytt värde både i push och i pop.

Vill man ha mer *generella och återanvändbara* funktioner för hantering av LIFO-listor skapar man sig en *abstrakt datatyp,* som innehåller data och funktioner för en allmän LIFO-lista. Man låter den arbeta på en allmän datatyp och varje användare får sedan infoga den aktuella datatypen, som ska gälla.

Ex: Skriv en abstrakt datatyp som exporterar data och operationer för en generell lista av LIFO-typ samt visa hur den används för att skapa en stack av reella tal.

```
/* Specifikation av LIFO-lista -- lifo.h */

typedef double datatyp; /* Om det nu var double vi ville ha. */

typedef
struct link
{
   datatyp data;
   struct link *next;
} linktyp;

void push(linktyp **lpp, datatyp d);
/* Stoppar in d i LIFO-listan */

datatyp pop(linktyp **lpp);
/* Tar bort data från LIFO-listan */



/* Implementation av LIFO-lista -- lifo.c */

#include ”lifo.h”
#include <stdlib.h>

void push(linktyp **lpp, datatyp d)
{
   linktyp *tp;

   tp = malloc(sizeof(linktyp));
   tp->data = d;
   tp->next = *lpp;
   *lpp = tp;
}

datatyp pop(linktyp **lpp)
{
   linktyp *tp;
   datatyp d;

   tp = *lpp;
   d = tp->data;
   *lpp = tp->next;
   free(tp);
   return d;
}
```

För att använda den abstrakta datatypen lifo, *måste man gå in i filen lifo.h* och med en typedef ge namnet datatyp till den typ av data, i vårt fall double, som ska stackas. Vill man inte att man ska gå in i filen lifo.h får man tillverka en anpassningsfil för data som man sedan alltid inkluderar i lifo.h eller använda void-pekare som datatyp.

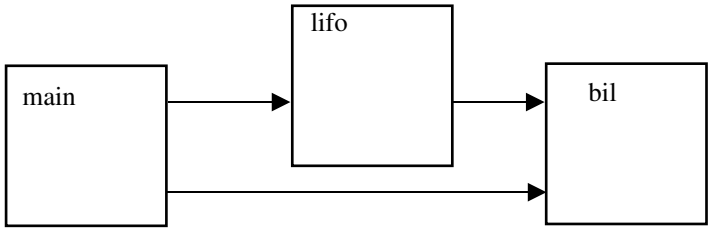
Ett huvudprogram som inkluderar lifo.h kan sedan utnyttja data och operationer för den aktuella LIFO-listan enligt:

```
/* Huvudprogram -- relifo.c */

#include <stdio.h>
#include ”lifo.h”

int main()
{
   linktyp *lp = NULL;
   double tal;

   printf("Ge tal : ");
   scanf("%lf", &tal);
   while (tal != 0.0)
   {
      push(&lp, tal);
      printf("Ge tal : ");
      scanf("%lf", &tal);
   }
   while (lp != NULL)
      printf("%f\n", pop(&lp));

   return 0;
}
```

I ovanstående exempel stackade man reella tal. Oftast är det poster som ska stackas och dessa poster kan i sin tur vara abstrakta datatyper innehållande både data och operationer, som avbildas med specifikation i en h-fil och implementation i en c-fil.

Det som är viktigt att komma ihåg är att man måste anpassa LIFO-listan för den aktuella datatypen genom att antingen gå in i lifo.h och inkludera den aktuella abstrakta datatypen och definiera om denna till aktuell typ eller också göra en anpassningsfil.

Ex: Skriv ett program som läser in data för bilar med registreringsnummer och ägare och stoppar in bilarna i en LIFO-lista, som sedan skrivs ut. Beroendet mellan de olika modulerna blir som :



Man implementerar oftast bottom-up dvs nerifrån och upp enligt:

```
/* Specifikation av bil - bil.h */

#ifndef BIL_H
#define BIL_H

typedef
struct bilpost
{
   char regnr[10];
   char agare[30];
} biltyp;

void read_bil(biltyp *bil);
/* Läs in data till bil från tangentbord */

void write_bil(biltyp bil);
/* Skriv ut data om bil på skärmen */

#endif


/* Implementation av bil - bil.c */

#include "bil.h"
#include <stdio.h>

void read_bil(biltyp *bil)
{
   printf("Ge bilens regnr : ");
   gets(bil->regnr);
   printf("Ge bilens ägare : ");
   gets(bil->agare);
}

void write_bil(biltyp bil)
{
   puts(bil.regnr);
   putchar('\n');
   puts(bil.agare);
   putchar('\n');
}
```

Sedan måste man in i lifo.h för att anpassa data för LIFO-listan till det aktuella datatypen enligt :

```
/* Specifikation av LIFO-lista -- lifo.h */


#include ”bil.h”          /* Här ändrar man */
typedef biltyp datatyp;   /* Här ändrar man */


typedef                      /* Resten som tidigare */
struct link
{
   datatyp data;
   struct link *next;
} linktyp;

void push(linktyp **lpp, datatyp d);
/* Stoppar in d i LIFO-listan */

datatyp pop(linktyp **lpp);
/* Tar bort data från LIFO-listan */
```

Till sist huvudprogrammet (och som omväxling använder vi lite Windows-specifika saker, med *conio.h*, *clearscr* och *getch*):

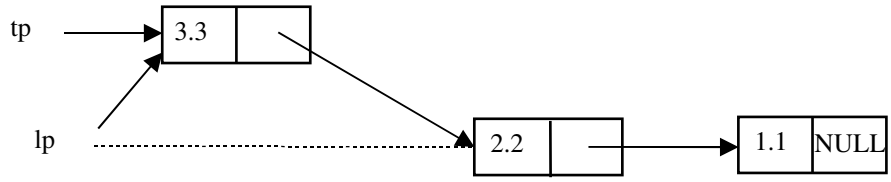
```
#include <stdio.h>
#include <conio.h>
#include "lifo.h"
#include "bil.h"

int main()
{
  linktyp *lp = NULL;
  biltyp bil;
  char ch;

  clrscr();
  printf("Ge bilar(tryck tangent, avsluta RETURN) : ");
  while ( getch() != '\r')
  {
     read_bil(&bil);
     push(&lp, bil);
     printf("Ge bilar(tryck tangent, avsluta RETURN) : ");
  }

  while (lp != NULL)
     write_bil(pop(&lp));

  getch();
  return 0;
}
```

3.1.2 FIFO-listor

En lista av FIFO-typ är en vanlig kö, där man som nytt objekt *ställer sig sist och den första i kön alltid blir betjänad*. Inga möjligheter ska finnas att bryta sig in i kön.

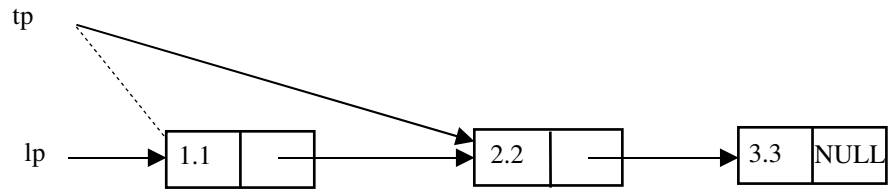
Man kan göra på samma sätt som i push och pop för LIFO-listan. Skillnaden är att när man stoppar in objekt i en FIFO *letar man sig till slutet och stoppar in objektet där istället för först*.

LIFO: Princip




```
tp = malloc(sizeof(linktyp));
tp->data = 3.3;
tp->next = lp;
lp = tp;
```

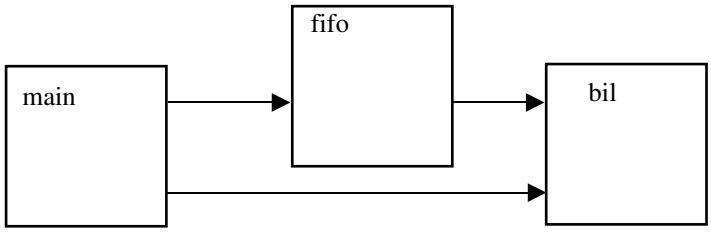
FIFO: Princip




```
tp = lp;
while (tp->next != NULL)
   tp = tp->next;
tp->next = malloc(sizeof(linktyp));
tp->next->data = 3.3;
tp->next->next = NULL;
```

Här måste man behandla en *tom lista* separat, eftersom man ej för en tom lista har någon next-pekare att testa på.

Ex: Skriv ett program som läser in data för bilar med registreringsnummer och ägare och stoppar in bilarna i en FIFO-lista som sedan skrivs ut. Beroendet mellan de olika modulerna blir som:




```
/* Specifikation av FIFO-lista -- fifo.h */

#ifndef FIFO_H
#define FIFO_H

#include "bil.h"            /* Exempelvis */
typedef biltyp datatyp;     /* Exempelvis */

typedef
struct link
{
   datatyp data;
   struct link *next;
} linktyp;

void infifo(linktyp **lpp, datatyp d);
/* Stoppar in d i FIFO-lista */

datatyp utfifo(linktyp **lpp);
/* Tar bort data från FIFO-listan */

#endif


/* Implementation av FIFO-lista -- fifo.c */

#include "fifo.h"
#include <stdlib.h>

datatyp utfifo(linktyp **lpp)
{
   linktyp *tp;
   datatyp d;

   tp = *lpp;
   d = tp->data;
   *lpp = tp->next;
   free(tp);
   return d;
}
```

```
void infifo(linktyp **lpp, datatyp d)
{
   linktyp *tp;

   if ( *lpp == NULL )
   {
      /* Första länken */
      *lpp = malloc(sizeof(linktyp));
      (*lpp)->data = d;
      (*lpp)->next = NULL;
   }
   else
   {
      /* Flytta fram tp att peka på sista länken */
      tp = *lpp;
      while ( tp->next != NULL)
         tp = tp->next;

      /* Stoppa in d efter sista länken i listan */
      tp->next = malloc(sizeof(linktyp));
      tp->next->data = d;
      tp->next->next = NULL;
   }
}



/* Huvudprogram - bilfifo.c */

#include "fifo.h"
#include "bil.h"
#include <stdio.h>
#include <conio.h>

int main()
{
  linktyp *fp = NULL;
  biltyp bil;

  clrscr();
  printf("Ge bilar(tryck tangent, avsluta RETURN) : ");
  while ( getch() != '\r')
  {
     read_bil(&bil);
     infifo(&fp, bil);
     printf("Ge bilar(tryck tangent, avsluta RETURN) : ");
  }

  while (fp != NULL)
     write_bil(utfifo(&fp));

  getch();
  return 0;
}
```

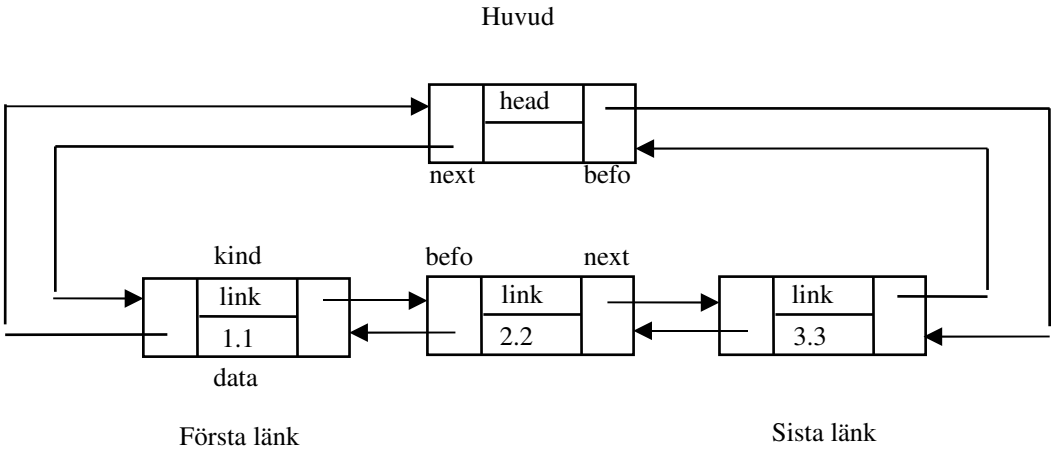
3.2 Tvåvägslistor

Nackdelen med envägslistor är att man bara kan genomlöpa dem i en riktning. Vill man ha en lista som man *kan genomlöpa i båda riktningarna* använder man en tvåvägslista som har två pekare i sina länkar. Den ena pekaren pekar på nästa länk och den andra pekar på föregående länk.

Själva länken ser ut som:

```
struct twolink
{
   enum {head, link} kind;
   struct twolink *next;
   struct twolink *befo;
   double data;
};
```

För att slippa problem som uppstår när man ska koppla ihop listan gör man den cirkulär och den cirkulära listans ingång markeras med en speciell länk, huvudet, som har värdet head på termen kind.



Man brukar avbilda en tvåvägslista som en abstrakt datatyp med en mängd operationer som ger intryck av att tvåvägslistan är rak med en första länk och en sista länk. I ovanstående figur kan man tänka sig att man delar på huvudet så att man får för sig att listan är rak även om den i verkligheten är cirkulär. Enligt detta sätt att se på en tvåvägslista har denna en början och ett slut.

Som vanligt delar man upp den abstrakta datatypen i en specifikations-fil twolist.h som visar vilka typer och operationer som exporteras samt en implementations-fil twolist.c, som visar funktionerna

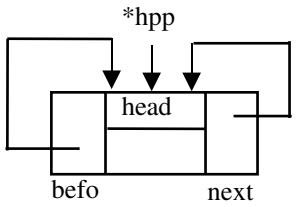
En tvåvägslista för reella tal kan man specifiera enligt:

```
/* Specifikation av tvåvägslista -- twolist.h */

#ifndef TWOLIST_H
#define TWOLIST_H


typedef double datatyp; /* Exempelvis om double ska användas */


typedef
struct twolink
{
   enum {head, link} kind;
   struct twolink *befo, *next;
   datatyp data;
} headtyp, linktyp;


void newhead(headtyp **hpp);
/* Skapar en ny tom lista */

void newlink(linktyp **lpp);
/* Skapar en ny tom länk */

void putlink(datatyp d, linktyp *lp);
/* Sätter in data i en länk */

datatyp getlink(linktyp *lp);
/* Returnerar data från länk */

void inlast(linktyp *lp, headtyp *hp);
/* Sätter in länken sist i listan */

void infirst(linktyp *lp, headtyp *hp);
/* Sätter in länken först i listan */

.. /* Resten av funktionsprototyperna, se nedan */

void clearhead(headtyp *hp);
/* Tar bort alla länkar från listan */

void elimhead(headtyp **hpp);
/* Eliminerar och NULL-ställer listan */

#endif
```

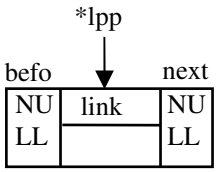
Ska man använda twolist för någon annan typ av data än reella tal får man ange detta i headerfilen ovan genom att *inkludera aktuell data och definiera om datatyp*.

Sedan följer implementationen av funktionerna enligt :

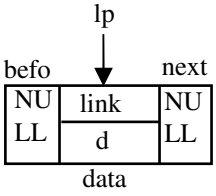
```
/* Implementation av tvåvägslistrutiner -- twolist.c */


#include <stdlib.h>
#include "twolist.h"


void newhead(headtyp **hpp)
/* Skapar en ny tom lista */
{
   *hpp = malloc(sizeof(headtyp));

   (*hpp)->kind = head;
   (*hpp)->befo = (*hpp)->next = *hpp;
}
```

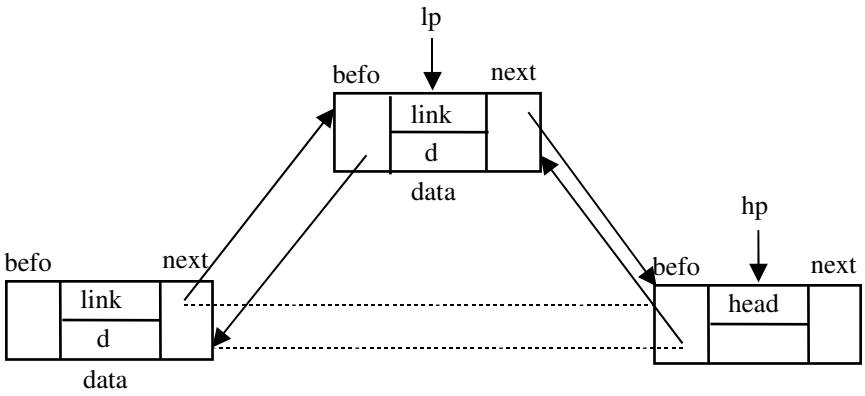



```
void newlink(linktyp **lpp)
/* Skapar en ny tom länk */
{
   *lpp = malloc(sizeof(linktyp));
   (*lpp)->kind = link;
   (*lpp)->befo = (*lpp)->next = NULL ;
}
```




```
void putlink(datatyp d, linktyp *lp)
/* Sätter in data i en länk */
{
   if (lp != NULL)
      lp->data = d;
}

datatyp getlink(linktyp *lp )
/* Returnerar data från en länk */
{
   datatyp d;

   if (lp != NULL)
      d = lp->data;
   return d;
}
```






```
void inlast(linktyp *lp, headtyp *hp)
/* Sätter in länk sist i lista */
{
   hp->befo->next = lp;
   lp->befo = hp->befo;
   hp->befo = lp;
   lp->next = hp;
}
```

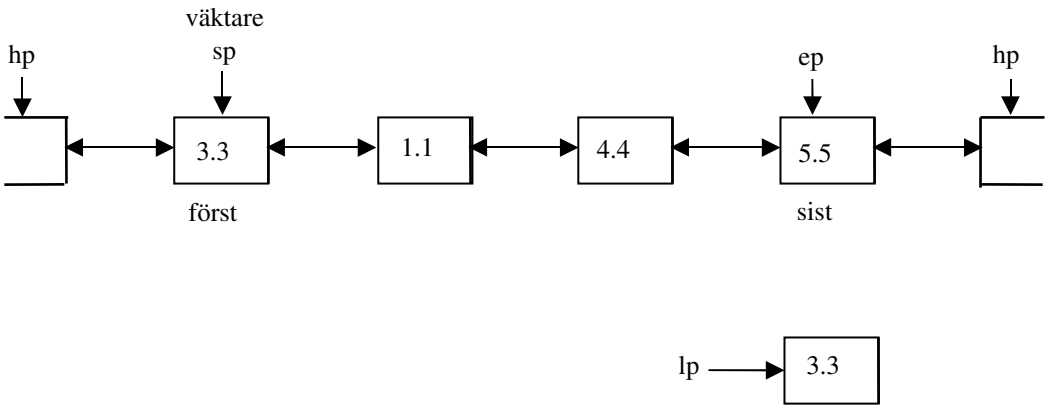
```
void infirst(linktyp *lp, headtyp *hp)
/* Sätter in länk först i lista */
{
   hp->next->befo = lp;
   lp->next = hp->next;
   hp->next = lp;
   lp->befo = hp;
}
```

```
void inpred(linktyp *lp, linktyp *ep)
/* Sätter in första länken före den andra */
{
   ep->befo->next = lp;
   lp->befo = ep->befo;
   ep->befo = lp;
   lp->next = ep;
}
```

```
void insucc(linktyp *lp, linktyp *ep)
/* Sätter in första länken efter den andra */
{
   ep->next->befo = lp;
   lp->next = ep->next;
   ep->next = lp;
   lp->befo = ep;
}
```

När man *sorterar* listor kan man naturligtvis använda de vanliga sorteringsalgoritmerna. Dessa är dock gjorda för datastrukturen vektor och man måste därför läsa från lista till vektor, sortera vektorn och sedan läsa tillbaka till listan, vilket verkar ganska omständigt och för stora listor också kräver en hel del minne.

Det är mycket bättre att se till att listan blir *sorterad direkt då man stoppar in data i den*. Man använder alltså instickssortering. För att samma algoritm även ska fungera för en tom lista kan man använda principen med en *väktarlänk* (”sentinel” på engelska), som utgör en kopia av den länk som ska instoppas och som man alltid stoppar in först i listan . När man gjort detta flyttar man sig längst bak i listan och jämför data för den länk, som ska instoppas med den sista länkens data. Ska länken stoppas in sist gör man detta. Annars flyttar man sig framåt i listan och gör en ny jämförelse osv.




```
void insort(linktyp *lp, headtyp *hp,
            int (*is_less)(datatyp d1, datatyp d2))
/* Sätter in länken sorterad enligt is_less */
{
   linktyp *sp, *ep;

   newlink(&sp);
   *sp = *lp;
   infirst(sp, hp);
   ep = lastlink(hp);
   while ( is_less(lp->data, ep->data) )
      ep = predlink(ep);
   insucc(lp, ep);
   elimlink(&sp);
}
```

```
linktyp *firstlink(headtyp *hp)
/* Returnerar pekare till första länken i listan */
{
   linktyp *ep;

   if ( !empty(hp) )
      ep = hp->next;
   else
      ep = NULL;
   return ep;
}



linktyp *lastlink(headtyp *hp)
/* Returnerar pekare till sista länken i listan */
{
   linktyp *ep;

   if ( !empty(hp) )
      ep = hp->befo;
   else
      ep = NULL;
   return ep;

}



linktyp *predlink(linktyp *lp)
/* Returnerar pekare till länken före  */
{
   linktyp *ep;

   if ( is_link(lp->befo) )
      ep = lp->befo;
   else
      ep = NULL;
   return ep;
}



linktyp *succlink(linktyp *lp)
/* Returnerar pekare till länken efter */
{
   linktyp *ep;

   if ( is_link(lp->next) )
      ep = lp->next;
   else
      ep = NULL;
   return ep;
}
```

```
int is_link(linktyp *lp)
/* Returnerar 1 om länk annars 0 */
{
   return (lp->kind == link);
}
```

```
int empty(headtyp *hp)
/* Returnerar 1 om listan tom annars 0 */
{
    return ( hp->next->kind == head );
}
```

```
int nrlinks(headtyp *hp)
/* Returnerar antalet länkar i listan */
{
   int sum = 0;
   linktyp *ep;

   ep = firstlink(hp);
   while ( ep != NULL )
   {
      sum++;
      ep = succlink(ep);
   }
   return sum;
}
```

```
void outlist(linktyp *lp)
/* Tar bort länken från listan */
{
   if (lp->befo != NULL && lp->next !=NULL)
   {
      lp->befo->next = lp->next;
      lp->next->befo = lp->befo;
      lp->befo = NULL;
      lp->next = NULL;
   }
}
```

```
void elimlink(linktyp **lpp)
/* Tar bort, avallokerar och NULL-ställer länken */
{
   outlist(*lpp);
   free(*lpp);
   *lpp = NULL;
}


void clearhead(headtyp *hp)
/* Eliminerar alla länkar från listan */
{
   linktyp *ep;

   while ( !empty(hp) )
   {
      ep = firstlink(hp);
      elimlink(&ep);
   }
}


void elimhead(headtyp **hpp)
/* Eliminerar och NULL-ställer listan */
{
   clearhead(*hpp);
   free(*hpp);
   *hpp = NULL;
}
```

Ovanstående 20 funktioner för att hantera tvåvägslistor kan man använda sig av i sina projekt för att hantera generella köer. Man måste in i headerfilen twolist.h och *inkludera aktuell data och definiera om denna till datatyp*. Sedan kan man kompilera upp en version av twolist som gäller för denna typ av data.

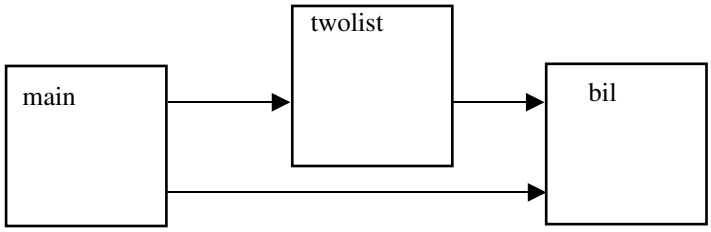
Alla funktioner i twolist blir *begränsade att gälla* för just den aktuella typen av data. Vill man i sitt program använda flera tvåvägslistor med *olika typer av data* kan man använda sig av en *union ingående i datatyp* eller också ha data *i form av void-pekare* i listan. Dessa void-pekare kan sedan sättas att peka på vilken typ av data som helst. Man kan ha blandade data i listan om man vill.

Säkerheten i ovanstående listrutiner är dålig. Färdiga biblioteksrutiner har betydligt bättre säkerhet. Man borde exempelvis sätta in *test på att det finns minne att allokera* i rutinerna newhead och newlink. Dessa kan göras om till int-funktioner och returnera 0 när det inte går att allokera. Man kan som användare öka säkerheten genom att testa på att länkpekaren är skilt ifrån NULL efter newlink.

Ex: Skriv ett huvudprogram, som använder funktionerna för tvåvägslistor för att skapa en lista av reella tal vilka läses in från tangentbordet, plockar bort alla negativa tal från listan och skriver ut den slutgiltiga summan av listans tal.

```
/* Huvudprogram -- rtwolist.c */

#include <stdio.h>
#include ”twolist.h”

int main()
{
   headtyp *hp = NULL;
   linktyp *lp = NULL, *ep = NULL;
   double tal, sum = 0;

   /* Skapa en tvåvägslista */
   newhead(&hp);
   printf("Ge tal : ");
   scanf("%lf", &tal);
   while (tal != 0.0)
   {
      newlink(&lp);
      putlink(tal, lp);
      inlast(lp, hp);
      printf("Ge tal : ");
      scanf("%lf", &tal);
   }

   /* Ta bort alla negativa tal */
   lp = firstlink(hp);
   while (lp != NULL)
   {
      if ( getlink(lp) < 0.0 )
      {
         ep = lp;
         lp = succlink(lp);
         elimlink(&ep);
      }
      else
         lp = succlink(lp);
   }

   /* Summera resterande tal */
   lp = firstlink(hp);
   while (lp != NULL)
   {
      sum += getlink(lp);
      lp = succlink(lp);
   }
   printf(”Summan = %f\n”, sum);
   return 0;
}
```

Ex: Skriv ett program som skapar en efter ägarnamn sorterad tvåvägslista av bilar med registreringsnummer och ägarnamn, frågar efter ett registreringsnummer och söker i listan efter denna bil, som skrivs ut på skärmen.




```
/* Specifikation av bil - bil.h */

#ifndef BIL_H
#define BIL_H

typedef
struct bilpost
{
   char regnr[10];
   char agare[30];
} biltyp;

void read_bil(biltyp *bil);
/* Läs in data från tangentbordet till bil */

void write_bil(biltyp bil);
/* Skriv ut data om bil på skärmen */

int less_agare(biltyp b1, biltyp b2);
/* Returnerar 1 om b1.agare < b2.agare */

#endif



/* Implementation av bil - bil.c */

#include "bil.h"
#include <stdio.h>
#include <string.h>

..

int less_agare(biltyp b1, biltyp b2)
{
   if (strcmp(b1.agare, b2.agare) < 0 )
      return 1;
   else
      return 0;
}
```

```
/* Specifikation av tvåvägslista - twolist.h */

#include ”bil.h”
typedef biltyp datatyp;

..



/* Huvudprogram -- btwolist.c */

#include <stdio.h>
#include <conio.h>
#include <string.h>
#include "twolist.h"
#include "bil.h"

int main()
{
  headtyp *hp = NULL;
  linktyp *lp = NULL;
  biltyp bil;
  char rstr[10];
  int found = 0;

  clrscr();
  newhead(&hp);
  printf("Ge bilar(tryck tangent, avsluta RETURN) : ");
  while ( getch() != '\r' )
  {
     read_bil(&bil);
     newlink(&lp);
     putlink(bil, lp);
     insort(lp, hp, less_agare);
     printf("Ge bilar(tryck tangent, avsluta RETURN) : ");
  }

  printf("Ge sökt regnr : ");
  gets(rstr);
  lp = firstlink(hp);
  while (lp != NULL && !found)
  {
     bil = getlink(lp);
     if ( strcmp(rstr, bil.regnr) == 0 )
        found = 1;
     else
        lp = succlink(lp);
  }

  if ( found )
     write_bil(bil);
  else
     printf("Finns ingen sådan bil!\n");
  getch();
  return 0;
}
```