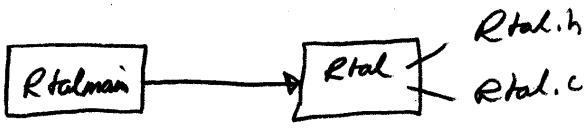
Repetition

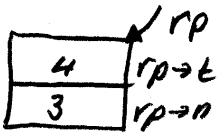
① **Abstrakta datatyper** – egendefinierade datatyper med tillhörande operationer

Ex Bråk




```
/*Rtal.h*/

#ifndef RTALH
#define RTALH

typedef struct
{  int t,n;
} rtal;

void las_rtal (rtal *rp);
void skriv_rtal (rtal r);
int minde_rtal (rtal r1, rtal r2);
int lika_rtal (rtal r1, rtal r2);
rtal add_rtal (rtal r1, rtal r2);
 |      |

#endif
```

```
/* Rtal.c */

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

void las_rtal(rtal *rp)
{
   printf("Ge ett bråk på formen a/b:");
   scanf("%d/%d", &rp->t, &rp->n);
}

void skriv_rtal(rtal r)
{
   printf("%d/%d\n", r.t, r.n);
}

int mindre_rtal(rtal r1, rtal r2)
{
   return  r1.t*r2.n < r2.t*r1.n;
}

int lika_rtal(rtal r1, rtal r2)
{
   return  r1.t*r2.t == r2.t*r1.n
}
;
void main()
{
   rtal a,b;

   las_rtal(&a);
   }
```



jämför

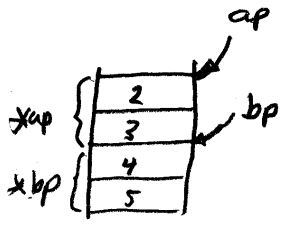
$\frac{2}{3} = \frac{2\cdot5}{3\cdot5} = \frac{10}{15}$

$\frac{4}{5} = \frac{4\cdot3}{5\cdot3} = \frac{12}{15}$

② Pekare och dynamisk allokering

(Ex) Skapa dynamiskt 2 bråk av typen rtal.
Läs in bråken och skriv ut i storleksordning

```
void main()
{
   rtal *ap, *bp;

   ap = malloc(sizeof(rtal));
   bp = malloc(sizeof(rtal));

   las_rtal(ap);
   las_rtal(bp);
   if (mindre_rtal(*ap, *bp))
   {  skriv_rtal(*ap);
      skriv_rtal(*bp);
   }
   else
   {  skriv_rtal(*bp);
      skriv_rtal(*ap);
   }
   }
   free(ap);
   free(bp);
   }
```



(Ex) Allokera dynamiskt en vektor med ett inläst antal bråk, Läs in bråken till vektorn och skriv ut i omvänd ordning.

```
void main()
{
    rtal *rv;
    int i, antal;

    printf("Hur många element? ");
    scanf("%d", &antal);
    rv = calloc(antal, sizeof(rtal));
    for(i=0; i<antal; i++)
    {
        las_rtal(&rv[i]);
    }
    for(i=antal-1; i>=0; i--)
    {
        skriv_rtal(rv[i]);
    }
    free(rv);
```

(3) Listor, Filer

(Ex) Läs alla bråk från textfilen Rtal.txt till stacken brakstack och skriv alla bråk som är lika stora som det sist inlästa.

4/5 1/3 2/3
1/4 1/5 9/27

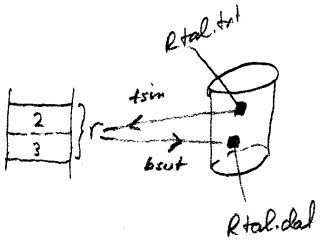
Anpassa Rtal.h till bråk!

```
void main()
{
   Linktyp *lp=NULL;
   rtal a,sista;
   FILE *tsin;
   tsin=fopen("Rtal.txt","rt");
   while(fscanf(tsin,"%d/%d",&a.t,&a.n) != EOF)
   {   push(&lp,a);
   }
   sista=a;
   {   while(lp != NULL)
       {
         a=pop(&lp);
         if (lika_sista(sista,a))
         {  skriv_rtal(a);
         }
       }
```

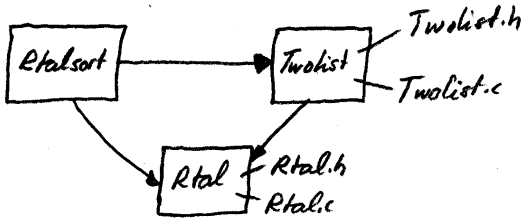
(Ex) Läs alla bråk från textfilen Rtal.txt till binärfilen Rtal.dat

1/4 2/3 1/3

```
void main()
{
   FILE *tsin, *bsut;
   rtal r;

  tsin = fopen("Rtal.txt", "rt");
  bsut = fopen("Rtal.dat", "wb");
  while(fscanf(tsin, "%d|%d", &r.t, &r.n) != EOF)
  {
     fwrite(&r, sizeof(rtal), 1, bsut);
  }
  fclose(tsin);
  fclose(bsut);
}
```

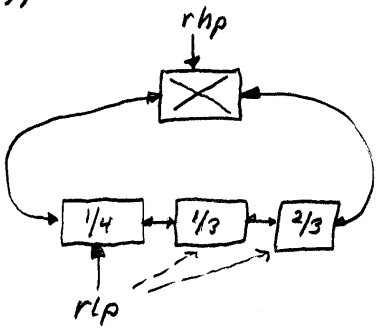


(Ex) Läs alla bråk från binärfilen Rtal.dat till en sorterad tvåvägslista och skriv ut listan.



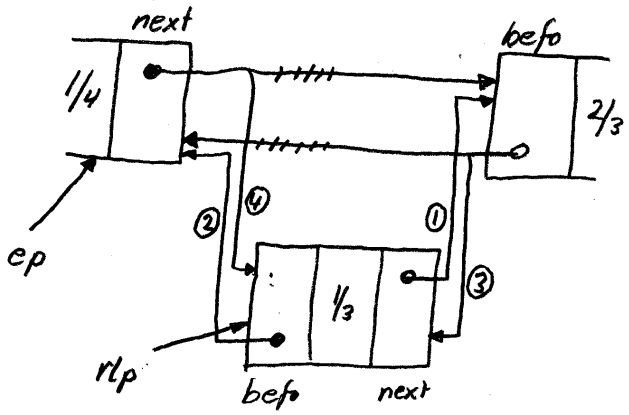

```
/* Twolist.h */
#include "Rtal.h"
typedef rtal datatyp;   } anpassa till aktuell data!

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

void newhead(headtyp **hpp);
!
void insort(linktyp *lp, headtyp *hp, int (*is_less)(datatyp,datatyp));
!
```

```
void main
{
   FILE *bsin;
   headtyp *rhp=NULL;
   Linktyp *rlp=NULL;
   rtal r;

   bsin=fopen("rtal.dat","rb");
   newhead(&rhp);
   fread(&r, sizeof(rtal), 1, bsin);
   while (!feof(bsin))
   {
        newlink(&rlp);
        putlink(r, rlp);
        insort(rlp, rhp, mindre_rtal);
   }
   fclose(bsin);
   rlp=firstlink(rhp);
   while (rlp!=NULL)
   {
        r=getlink(rlp);
        skriv_rtal(r);
        rlp=succlink(rlp);
   }
   elimhead(&rhp);
}
```



Pekaromställningar – ändra adresser

(Ex) Hur gör insort?



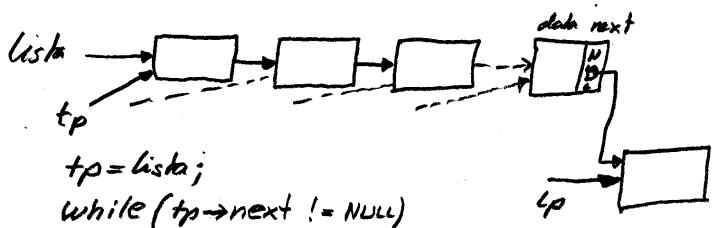
① `rlp->next = ep->next;`

② `rlp->befo = ep;`

③ `ep->next->befo = rlp;`

④ `ep->next = rlp;`

(Ex) Sätt in länken lp sist i lista.




```
tp = lista;
while (tp->next != NULL)
{
   tp = tp->next;
}
tp->next = lp;
lp->next = NULL;
```

* eller & eller inget?

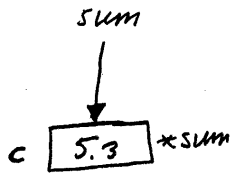
(Ex) En summafunktion

```
void summa(float x, float y, float *sum)
{
    *sum = x+y;
}
void main()
{
    float a=3.5, b=1.8, c;
    summa(a, b, &c);
    printf("Summan = %.1f", c);
}
```

datatyp

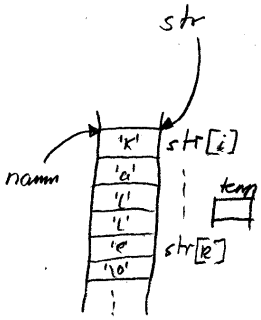
*sum: avreferens

&c: adress



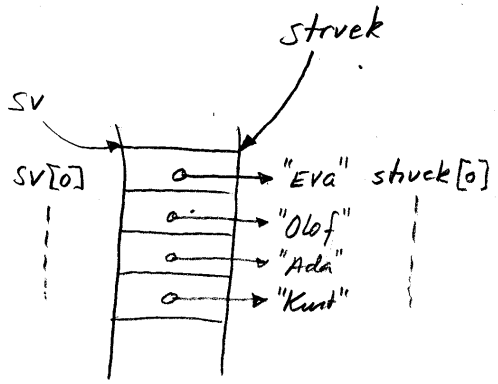
(Ex) En baklängesfunktion

```
void bak(char *str)
{
    int i, k;
    char ch;
    for(i=0, k=strlen(str)-1; i<k; i++, k--)
    {
        temp = str[i];
        str[i] = str[k];
        str[k] = temp;
    }
}
void main()
{
    char namn[20]="kalle";
    bak(namn);
    :
```



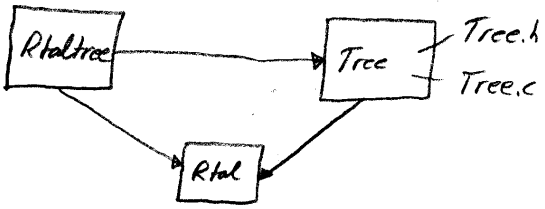
(Ex) Vektor av strängar.

```
void alla_bak (char *strvek[], int nr)
{
    int i;

    for(i=0; i<nr; i++)
    {
        bak(strvek[i]);
    }
}

void main()
{
    char *sv[4]={"Eva", "Olof", "Ada", "Kurt"};

    alla_bak(sv, 4);
    :
}
```



Sökning

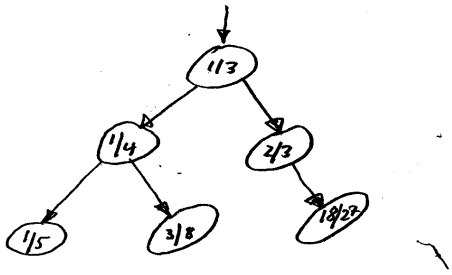
(Ex) Läs in alla bråk från Rtal.dat till ett binärt träd och sök efter ett inläst bråk.




```
/* Tree.h */

#include "Rtal.h"
typedef rtal datatyp }  anpassa till aktuell data

typedef
struct nod
{
   datatyp data;
   struct nodtyp *left, *right;
} nodtyp;

void intotree (nodtyp **npp, datatyp d,
                int (*is_less)(datatyp, datatyp));

nodtyp *searchtree (nodtyp *np, datatyp key,
                    int (*is_less)(datatyp, datatyp),
                    int (*is_equal)(datatyp, datatyp));
```

```
/* Rtaltree.c */
#include "Rtal.h"
#include "Tree.h"

void main()
{
    FILE *bsin;
    rtal r, nyckel;
    nodtyp *rnp = NULL;

    bsin = fopen("Rtal.dat", "rb");
    fread(&r, sizeof(rtal), 1, bsin);
    while (!feof(bsin))
    {
        intotree(&rnp, r);
        fread(&r, sizeof(rtal), 1, bsin);
    }
    fclose(bsin);
    Los_rtal(&nyckel);
    if (searchtree(rnp, nyckel, mindre_rtal, lika_rtal) == NULL)
    {
        printf("Bråket finns ej!");
    }
    else
    {
        printf("Bråket finns!");
    }
}
```

(Ex) Skriv en rekursiv funktion som beräknar antalet lika stora bråk i ett träd.




```
int nr_lika (nodtyp *np, datatyp key, int (*is_less)
             (datatyp, datatyp), int (*is_equ(datatyp, datatyp))
{
    if (np == NULL)
    {   return 0;
    }
    else if (is_equal (key, np->data))
    {
        return 1 + nr_lika (np->right, key, is_less, is_equal);
    }
    else if (is_less(key, np->data))
    {   return nr_lika(np->left, key, is_less, is_equal);
    }
    else
    {
        return nr_lika (np->right, key, is_less, is_equal);
    }
}
```