Rekursive datatyper

et naturlig tall m er

1. null eller : 0 eller
2. en etterfølger av et naturlig tall n : 1 + n

en sekvens av ‘element’ er

1. en tom sekvens eller : sekvens = Ø eller
2. *et element* etterfulgt av en **sekvens** : sekvens = *element* & **sekvens**

head **tail**

Man kan da definere operasjonene rekursivt :

1. angi hva som skal gjøres i basistilfelle (termineringstilfelle)
2. ellers angi hva som skal gjøres
   -med elementet (*head*)
   -før (ev. etter) man fortsetter med resten av sekvensen (**tail**)

```
OP ( sek ) =
1. if sek = Ø then <basisltilfelle - ev. andre termineringsbetingelser>
2. else
     begin
       < gjør jobben med head >
      OP ( sek . tail ) ;  < fortsett med tail - rekursivt kall >
     end ;
```


Rekursive operasjoner

et naturlig tall

1. m = 0 eller
2. m = 1 + n , der n er et naturlig tall

```
s + m =
1. if m = 0 then s
2. else (1+s) + n                    ! her er m = 1+n ;
```

en sekvens er

1. sek = Ø eller
2. sek = *element* & **tail**-sekvensen

```
lengden (sek ) =
1. if sek = Ø then  0
2. else
     begin
      1 +                       < gjør jobben med head >
     lengden ( sek . tail ) ;  < fortsett med tail - rekursivt kall >
     end ;
```

tail recursion

```
s :- sek;
l := 0;
while s <> ø do
begin
   l := l + 1 ;
   s :- s . tail ;
end
```


Tabell er en sekvens

en tabell (med heltall) er

1. tom eller : tabell = Ø eller
2. et tall etterfulgt av resten av tabellen: tabell = *tall* + **resten**

```
SORTER ( T )
  if T <> Ø then
   begin
     i := < indeksen til minste tallet i T >
     < bytt tallene T(1) og T(i) >
     SORTER (resten) ;
   end;
```

sekvensen *sek* er tom hviss sek . head == **none**

```
class sequence ;
   ref(element) head;   ref(sequence) tail;
. . .
   procedure sort ;
   if  head =/= none then
   begin ref(sequence) i ; ref(element) e ;
      i :- finn_minste (this sequence) ;
      e :- i . head ;   i . head :- head ;   head :- e ;
      if tail =/= none then  tail . sort ;
   end;

   ref(sequence) procedure finn_minste(S); ref(sequence) S;
     ! S er sekvensen med minste ‘head’ ved kalltidspunktet ;
   begin ref(sequence) h;
      h :- if head == none or else head >= S. head then S else this sequence ;
      finn_minste :- if tail == none then h else  tail . finn_minste(h)  ;
   end ;
```


Rekursiv implementasjon av sekvens

sekvensen *sek* er tom hviss sek . head == **none**

```
class element(key); text key; ;
class sequence;
begin
   ref(element) head;   ref(sequence) tail;

   procedure append(el); ref(element) el;
   if head == none then head :- el
   else    begin
             if tail == none then tail :- new sequence ;
             tail . append(el);
   end;

   integer procedure size ;
   size :=  if head == none then 0
             else if tail == none then 1
             else 1 +  tail . size  ;

   ref(element) procedure find(id); text id;
   find :-   if head =/= none and then  head . key = id then head
             else if tail == none then none
             else   tail . find (id)  ;

   procedure remove(id); text id;
   if head =/= none and then  head . key = id then
   begin
      if tail == none then head :- none
      else begin  head :- tail . head;   tail :- tail . tail;   end;
   end else
      if tail =/= none then   tail . remove(id)  ;

   procedure for_each(p); procedure p;
   if head =/= none then
     begin
      p (head) ;
      if tail =/= none then  tail . for_each(p)  ;
     end;
end;
```

```
ref(sequence) sek;
sek :- new sequence ;
...
sek . append (el);
sek . size ;
sek . find(“aa”) ;
...
```

sequence class ...

```
class sequence;
begin
    ref(element) head;    ref(sequence) tail;
    procedure append(el); ref(element) el;
    integer procedure size ;
    ref(element) procedure find(id); text id;
    procedure remove(id); text id;
    procedure for_each(p); procedure p;
end;

sequence class LIFO;
begin
    procedure remove_L;
        if tail == none then head :- none
        else if tail . head == none then head :- none
        else  tail . remove_L ;
end;

sequence class FIFO;
begin
    procedure remove_F;
        if tail == none then head :- none
        else begin
            head :- tail . head ;
            tail :- tail . tail ;
        end ;
end;
```


Trær

et tre av 'element' er

1. et tomt tre   eller   : <u>tre</u> = Ø   eller
2. en *node* (et *element*) med   : <u>tre</u> = *element* + **<u>tre1</u>** + **<u>tre2</u>** + .. + **<u>trex</u>**
   x disjunkte **<u>subtrær</u>**

I et binært tre har anhver node høst 2 subtrær: venstre og høyre.

```
class element(id); integer id ; ;

class B_Tre;
begin
    ref(element) data;    ref(B_Tre) left, right;
    procedure add(el); ref(element) el;            ! f. eks. ;
        if data == none then data :- el
        else if left == none then
            begin  left :- new B_Tre;  left . add(el) ;  end
        else
            begin if right == none then right :- new B_Tre;  right . add(el) ; end;
    integer procedure size ;
    size :=  if data == none then 0
             else  1  + (if left == none then 0 else  left . size )
                      + (if right == none then 0 else  right . size ) ;
    ref(element) procedure find(id); integer id;
        if data == none then find :- none
        else if data . id = id then find :- data
        else begin ref(element) h;
            if left =/= none then h :-  left . find(id) ;
            if h == none and then right =/= none then h :-  right . find(id)  ;
            find :- h ;
        end ;
```

et tre *T* er tomt
hviss
T . data **==** **none**

Settools "implementasjon" av Trær

```
class dta(id); integer id ; ;

element class B_Tre;
begin
    ref(dta) data;    ref(Set) sub;

    text procedure key; ! .... ? ....  - bruk ref(Sequence) sub isteden ;

    procedure add(d); ref(dta) d;          ! trenger "indeks" der d skal settes inn;

    integer procedure size ;
        if data == none then size := 0
        else   ~~size := 1 + sub . for_each_element (size) ;~~
        begin ref(B_Tre) h; integer i ;
            h :- sub . first element; i := 1 ;
            while h =/= none do
                begin
                  i := i + h . size;
                  h :- sub . next_element;
                end ;
            size := i;
        end ;
    ref(dta) procedure find(id); integer id;
        if data == none then find :- none
        else if data . id = id then find :- data
        else begin ref(B_Tre) h; boolean OK ; ref(dta) d;
            h :- sub . first element;
            while h =/= none and not OK do
                begin
                  d :- h . find(id);
                  if d =/= none then OK:= true else h :- sub . next_element;
                end ;
            find :- d;
        end ;
```