Et eksempel (Julenissen)

Et barn kan være snilt eller slemt på en skala fra 0 og oppover, der 0 er veldig, veldig snill. Vi merker av et skille (S) på skalaen, og regner alle barn med slemhetsverdi mindre eller lik S som snille, og alle de andre som slemme. Skillet S kan gå hvor som helst, og kan settes forskjellig hvert år. Det kan være vilkårlig mange snille og slemme barn. Hvert barn har et unikt navn.

Så kommer julen, og Julenissen med en sekk med gaver. Hver gave har et entydig navn. Han gir gaver: han spør hvert snilt barn hvilken gave barnet vil ha. Han spør først det snilleste barnet, så det nest snilleste og så videre. Hvert av de snille barna kan velge en gave som fjernes fra Julenissens sekk og gis til barnet. Deretter skal alle slemme barn få en vilkårlig gave blant de som er igjen i sekken. Julenissen har nok gaver til at hvert barn får en.

DIN JOBB er å lage Julenisseprogrammet.

Lag en datastruktur for å holde rede på alle barna og gaver.
Du kan anta at det fins en

text procedure sort_int_as_text(i) ; **integer** i ;

som konverterer et heltall til en tekst på en slik måte at, dersom i < j som tall så vil også sort_int_as_text(i) < sort_int_as_text(j) som tekst.

Lag grove skisser for følgende prosedyrer:

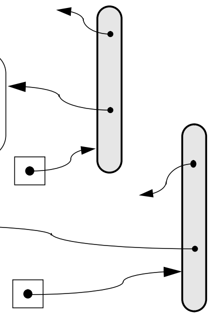
1. Nytt_barn – som lager et nytt barn

2. Gi_gaver – som (a) spør alle de snille barna, i riktig rekkefølge, hvilken gave de vil ha, fjerner gaven fra sekken og gir den til barnet og (b) deretter velger en og en gave fra sekken og gir den til ett og ett slemt barn (i vilkårlig rekkefølge).

3. Skriv – som skriver på skjermen: først alle de snille barna (navn), og deretter alle de slemme barna, samt gave for hvert barn.

Julenissen

```
begin   external class Settools;
Settools begin
  element class barn ;
  begin
     text navn;  boolean snilt;  integer hvor_slemt;
     text procedure key;
       key :– sort_int_as_text(hvor_slemt)&” “&navn;
     ref(gave) min_gave;
  end;
  ref(Ordered_Set) barna;
  element class gave;
  begin
     text navn;
     text procedure key; key :– navn;
  end;
  ref(Set) sekk;
  procedure Nytt_barn;
   begin  ref(barn) b;
      b :– new barn;  < les barnet >;   barna . add_element_ok(b);
   end ;
  procedure Gi_gaver;
   begin  ref(barn) b;  ref(gave) g; text gave_navn;
   b :– barna . first_element;
   while b =/= none and then b . snilt do
     begin
       gave_navn :– <spør barnet hva det vil ha > ;
       g :– sekk . find_element(gave_navn);
       if g =/= none then begin
          b.min_gave:–g ;  sekk.remove_element_ok(gave_navn);
          b :– barna . next_element;
       end else  < Skriv at gaven ikke finns >;
     end;
   while b =/= none do
     begin
       g :– sekk . first_element;
       b . min_gave :– g ;   sekk . remove_element_ok( g . navn );
       b :– barna . next_element;
     end;
   end Gi_gaver;       . . . . .
end  Settools;
end
```



for hvert snilt barn:
gi det gaven det velger

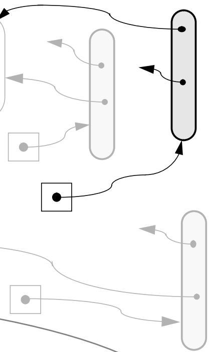
for hvert slemt barn:
gi det en vilkårlig gave

En mengde for hver kategori

(for_each_element(p); **procedure** p;)

```
element class barn ;
 begin
    text navn;   boolean snilt;  integer hvor_slemt;
    text procedure key;
      key :– sort_int_as_text(hvor_slemt)&” “&navn;
    ref(gave) min_gave;
 end;
ref(Ordered_Set) snille;
ref(Set) slemme;
element class gave;
 begin
    text navn;
    text procedure key; key :– navn;
 end;
ref(Set) sekk;

procedure Nytt_barn;
 begin  ref(barn) b;
    b :– new barn;  < les barnet >;
    if <b er snilt> then snille . add_element_ok(b)
                   else slemme. add_element_ok(b) ;
 end ;
procedure Gi_gaver;
   begin
    snille . for_each_element ( Gi_snilt) ;
    slemme. for_each_element ( Gi_slemt) ;
   end;
procedure Gi_snilt(b); ref(barn) b;
 begin text gave_navn; ref(gave) g;
    gave_navn :– <spør barnet hva det vil ha > ;
    g :– sekk . find_element(gave_navn);
    if g =/= none then begin
       b . min_gave :– g ;  sekk . remove_element_ok(gave_navn);
    end else begin < Skriv at gaven ikke fins >; Gi_snilt(b); end;
 end;
procedure Gi_slemt(b); ref(barn) b;
   begin  ref(gave) g;
    g :– sekk . first_element;
    b . min_gave :– g ;   sekk . remove_element_ok( g . navn );
   end;
```

```
b :– snille . first_element;
while b =/= none do
 begin
   Gi_snilt(b);
   b :– snille.next_element;
 end;
b :– slemme . first_element;
while b =/= none do
 begin
   Gi_slemt(b);
   b :– slemme.next_element;
 end;
```



En mengde kan inneholde forskjellige element’er

```
element class barn ;
 begin
    text navn;
    integer hvor_slemt;
    text procedure key;
      key :– sort_int_as_text(hvor_slemt)&” “&navn;
    ref(gave) min_gave;
 end;
ref(Ordered_Set) barna;
element class gave;
 begin
    text navn;
    text procedure key; key :– navn;
 end;
ref(Set) sekk;
ref(barn) b1, b2;
ref(gave) g1, g2;

barna :– new Set;
b1 :– new barn; b2 :– new barn;
g1 :– new gave; g2 :– new gave;
. . . .
barna . add_element_ok(b1);
barna . add_element_ok(g2);
barna . add_element_ok(g1);
```

Mengder vs. Tabeller

```
element class ting;
  begin text procedure key; . . end;
ref(ting) g;

ref(Sequence) seq;

seq . append (new ting) ;

g :– seq . element_number(k) ;
```

```
class ting; ;
ref(ting) g;

ref(gave) array tab(1:n) ;
integer n, i ;

if i <= n then tab(i) :– new ting ;
i := i+1;

g :– tab(k) ;
```

```
element class ting(i,j); integer i, j;
begin
  text procedure key; key :– ij(i,j);
end;

text procedure ij(i,j); integer i, j;
  ij:–int_as_text(i)&" "&int_as_text(j);

ref(Ordered_Set) os;
os :– new Ordered_Set;

os . add_element_ok(new ting(i,j));

g :– os . find_element( ij(i,j) );
```

```
ref(ting) array tab(1:n, 1:m) ;
integer n, m, i, j ;

tab(i, j) :– new ting ;

g :– tab(i, j) ;
```

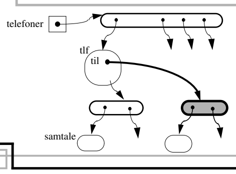
```
class tabell(n,m); integer n, m;
begin
  ref(ting) array b(1:n, 1:m);
end;

ta :– new tabell(20, 30) ;

ta . b(i, j) :– new ting ;

g :– ta . b(i, j) ;
```


	Mengder	Tabeller
elementene identifiseres	ved en (meningsfylt) tekst-nøkkel	ved sin posisjon blant andre elementer
antall elementer	ukjent og vilkårlig	"kjent" på forhånd
plassering av elementene	spiller ingen rolle hvor som helst (på slutten, i alfabetisk rekkefølge)	et bestemt sted i helheten
referanse til et element (oppslag)	ved en nøkkel-verdi (søking - mindre effektiv)	ved en posisjon (direkte - meget effektiv)
Brukes når	– begrensede strukturen er tilstrekkelig for det aktuelle problemet – det er lett (mulig) å identifisere elementene ved et nøkkel-attributt – strukturen er forholdsvis dynamisk: hyppige innsettinger/fjerninger	– problemeområdet er strukturert – plassering av elementene i datastrukturen er avgjørende – deres identitet spiller mindre rolle – strukturen er statisk: hyppige oppslag

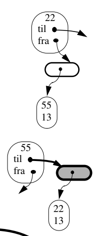
Dobbelt lagring av data

For hver telefon (med et entydig nummer) skal programmet kunne skrive følgende:
a) alle telefonnummrene det ble ringt til (fra denne telefonen),
b) alle telefonnummrene det ble ringt fra (til denne telefonen),
c) for hvert av disse (i a og b) totalt antall tellerskritt.

```
ref(Set) telefoner;

element class tlf(nr);  integer nr;
  begin
    text procedure key; key :– int_as_text(nr) ;
    ref(Ordered_Set) til ;
    ref(Ordered_Set) fra ;
    fra :– new Ordered_Set;
    til :– new Ordered_Set;
  end;

element class samtale(nr);  integer nr;
  begin
    integer tskritt;
    text procedure key; key :– int_as_text(nr) ;
  end;
```




```
procedure Ny_samtale(fra,til,s);  integer fra, til, s;
  begin  ref(tlf) t; ref(samtale) sam;
    t :– telefoner . find_element( int_as_text(fra) );
    sam :– t . fra . find_element( int_as_text(til) );
    if sam == none then
     begin
      sam :– new samtale(til);
      t . fra . add_element_ok(sam);
     end;
    sam . tskritt := sam . tskritt + s ;
    t :– telefoner . find_element( int_as_text(til) );
    sam :– t .til . find_element( int_as_text(fra) );
    if sam == none then
     begin
      sam :– new samtale(fra);
      t .til . add_element_ok(sam);
     end;
    sam . tskritt := sam . tskritt + s ;
  end ;

procedure Skriv(nr);  integer nr;
  begin  ref(tlf) t;
    t :– telefoner . find_element( int_as_text(nr) );
    < for hver samtale i t . fra – skriv dataene >
    < for hver samtale i t . til – skriv dataene >
  end ;
```



samtale(T2.nr,s) i T1.fra hvis og bare hvis samtale(T1.nr,s) i T2.til

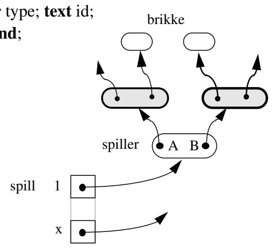
Lagring (sortering) etter forskjellige nøkler

Hver person har et navn og entydig personnummer. En bedrift skal lagre data om alle ansatte slik at man skal kunne se en liste sortert etter
a) personnummer
b) navn (og deretter personnummer)
c) lønn (og deretter navn).

```
element class person(navn, pnr); text navn; integer pnr;
  begin
    text procedure key; key :– int_as_text(pnr) ;   ! sort_int_as_text(pnr) ;
    integer lonn ;
  end;

ref(Ordered_Set) ansatte;

element class skygge(p, sort); ref(person) p; text sort;
  begin
    text procedure key; key:– if sort="NP" then NP(p) else LN(p);
  end;

text procedure NP(p); ref(person) p; NP :– p . navn &" "& p . key ;

text procedure LN(p); ref(person) p;
  LN :– sort_int_as_text(p . lonn) &" "& p . navn &" "& p . key ;

procedure Skriv_ansatte  (sort); text sort;
  if sort = "P" then    ansatte . display
  else begin ref(Ordered_Set) tmp; ref(person) p;
    tmp :– new Ordered_Set;
    p :– ansatte . first_element;
    while p =/= none do
      begin
        tmp . add_element_ok(new skygge(p,sort));
        p :– ansatte . next_element;
      end;
    tmp . display ;
  end;
```

Når navn ikke er viktig, men kan være nyttig . . .

Et spill kan spilles av 2 til x spillere, der hver spiller begynner med n brikker av type A og n brikker av type B.




```
integer x;
x := inint ;
  begin
  element class brikke(type, id); character type; text id;
     begin text procedure key; key :– id; end;

  class spiller ;
     begin
       ref(Set) A, B;
       A :– new Set; B :– new Set;
     end;

  ref(spiller) array spill(1 : x);

  procedure fordel_brikker(n); integer n;
    begin integer s, b;
    for s:=1 step 1 until x do
      begin
        spill(s) :– new spiller; b:=1;
        while b <= n do
          begin
            spill(s) . A . add_element_ok(
                        new brikke(‘A’, int_as_text(s) & “A” & int_as_text(b) )) ;
            spill(s) . B . add_element_ok(
                        new brikke(‘B’, int_as_text(s) & “B” & int_as_text(b) )) ;
            b:=b+1;
          end ;
      end;
    end fordel_brikker;
  < Spill >
  end ;
```


Et lite problem ....

```
begin
external class settools;
Settools begin

  element class brikke(id); text id;
    begin text procedure key; key :– id; end;
  ref(Set) alle ;
  ref(brikke) b1, b2, b3, b ;

  alle :- new Set ;
  b1 :- new brikke(“aaa”) ;
  b2 :- new brikke(“bbb”) ;
  b3 :- new brikke(“aaa”) ;

  alle . add_element_ok(b1) ;
  alle . add_element_ok(b2) ;
       ! alle = { b1, b2 } ;
  alle . add_element_ok(b3) ;
       ! alle = { b1, b2 } , siden b3 . key = b1 . key ;
  b :- alle . find_element(“bbb”) ;
       ! b == b2 ;

  b1 . id :- “bbb” ;
       ! alle = { b1, b2 } , men b1 . key = “bbb” og b2 . key = “bbb” ;
  b :- alle . find_element(“bbb”) ;
       ! b == b1 ;
```