Runtimesystemer - II

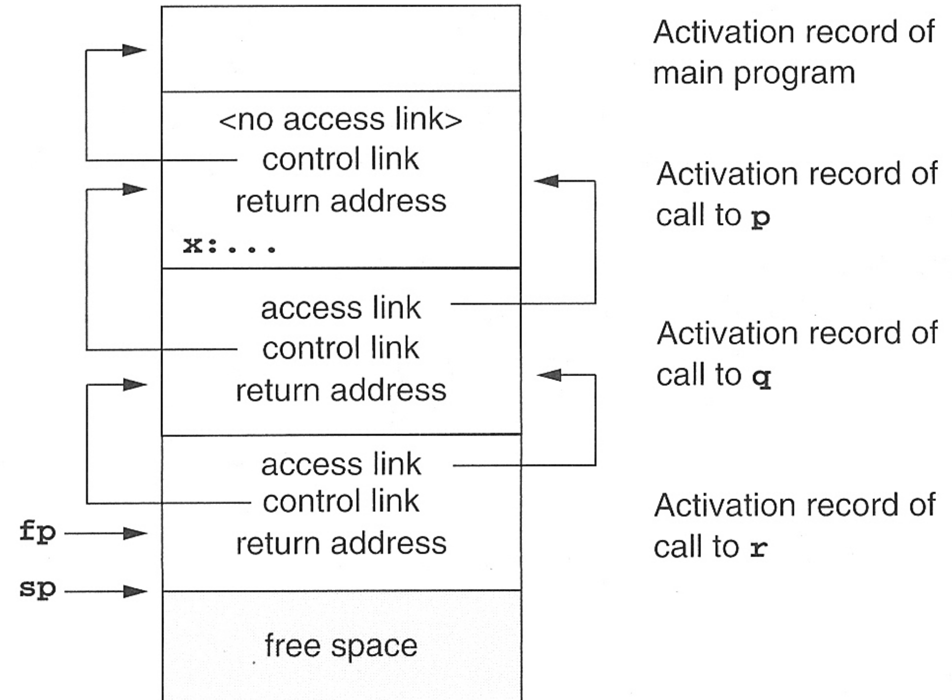
- Funksjoner som parametere
- Virtuelle metoder
- Parameteroverføring
  - Call by value
  - Call by reference
  - Call by value-result
  - Call by name

FUNKSJONER SOM PARAMETERE

Eksempel med flere nivåer

```
program chain;

procedure p;
var x: integer;

   procedure q;
      procedure r;
      begin
         x := 2;
         ...
         if ... then p;
      end; (* r *)
   begin
     r;
   end; (* q *)

begin
  q;
end; (* p *)

begin (* main *)
  p;
end.
```

Program-blokkene får da et blokk-nivå



display

0	
1	
2	
	.

fp.al.al.x

diff i blokknivå

Prosedyrer som parametere

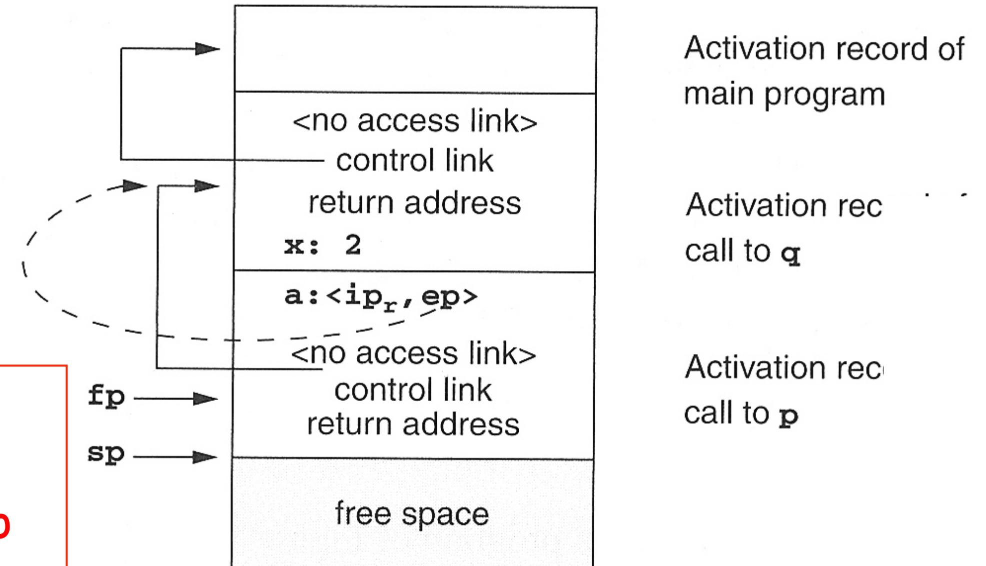
```
program closureEx(output);

procedure p(procedure a);
begin
  a;
end;

procedure q;
var x:integer;

      procedure r;
      begin
        writeln(x);
      end;

begin
  x := 2;
  p(r);
end; (* q *)

begin (* main *)
  q;
end.
```

Representeres ved en closure: (ep,ip)

Dette må da oversettes slik:

1. aksess-peker = ep
2. hopp til ip

ip_r
ep_r

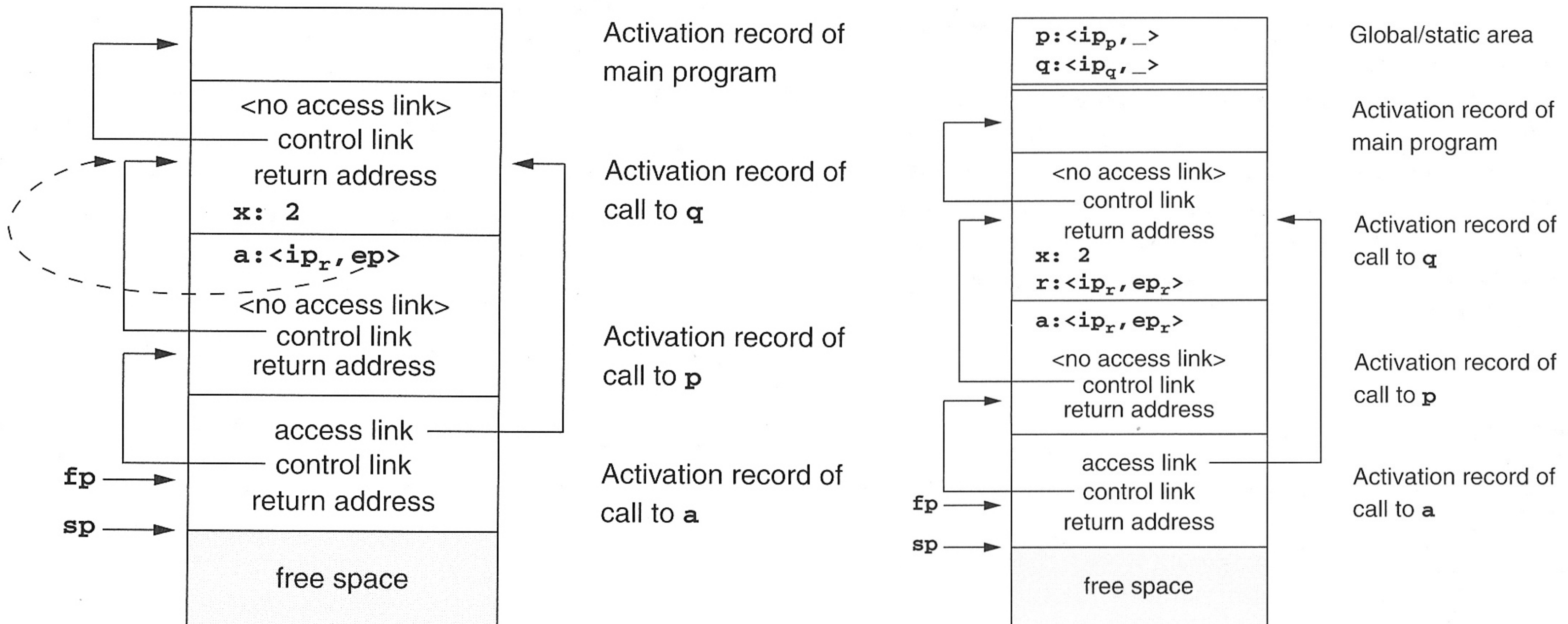


Den aktuelle parameteren må være:

- Kode-adressen til prosedyren (ip)
- Prosedyrens aksess-link (ep)

Kall av prosedyre levert som parameter

- Etter kallet på den formelle parameteren ’a’ som aktuelt er ’r’ i Q:



INF 5110 - 2014

Funksjon/..., eksempel

```
program
   var pv: procedure (integer);

   procedure Q();
      var a: integer;
      procedure P(integer i)
         a := a + i;
      end;
      ...
      pv := p
      ...
   end;
   ...
   Q();
   pv(3);
end
```

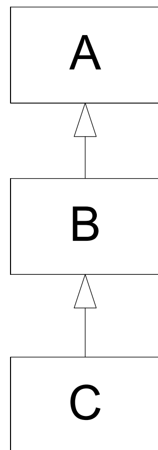
'a' finnes ikke

Men:
Prosedyrer som parametre til prosedyrer gir ingen slike problemer

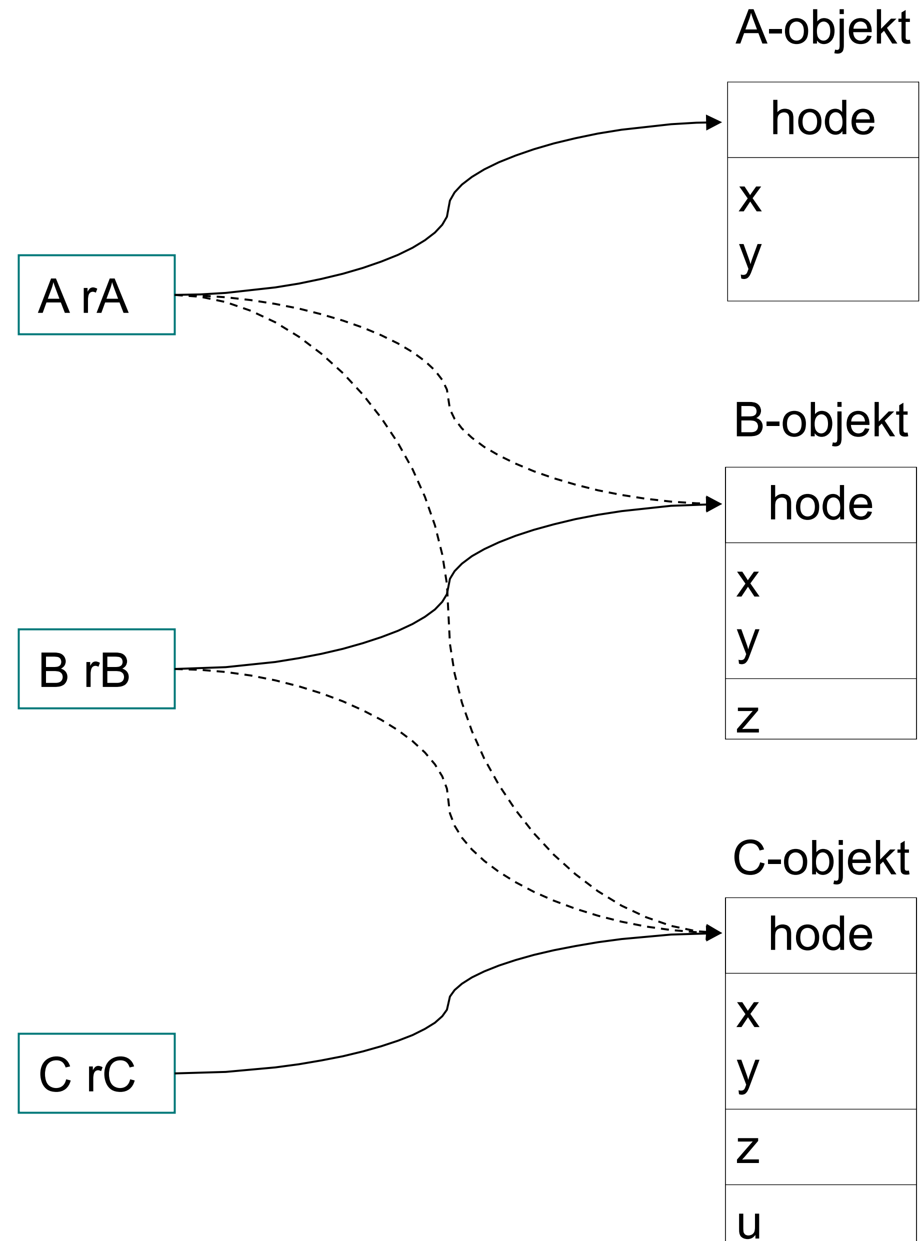
VIRTUELLE METODER

Objekt-orientering

- Klasser og subklasser
- Typede pekere
- Virtuelle og ikke-virtuelle metoder




```
class A {
  int x,y;
  void f(s,t) {...K...};
  virtual void g(p,q) {...L...}
}

class B extends A{
  int z;
  void f(s,t) {....Q...};
  redef void g(p,q) {...M...};
  virtual void h(r) {...N...}
}

class C extends B{
  int u;
  redef void h(r) {...P...}
}
```



- Kall på ikke-virtuelle metoder (bruk pekerens type)
  - rA.f(1,2) (K)
  - rB.f(1,2) (Q)
  - rC.f(1,2) (Q)
- Kall på virtuelle metoder (bruk objektets type)
  - rA.g(3,4) (L eller M)
  - rB.g(3,4) (M)
  - rC.g(3,4) (M)
  - rA.h(5) ulovlig
  - rB.h(5) (N eller P)
  - rC.h(5) (P)

```
class A {
  int x,y;
  void f(s,t) {...K...};
  virtual void g(p,q) {...L...}
}

class B extends A{
  int z;
  void f(s,t) {....Q...};
  redef void g(p,q) {...M...};
  virtual void h(r) {...N...}
}

class C extends B{
  int u;
  redef void h(r) {...P...}
}
```

Implementasjon: virtuell-tabell

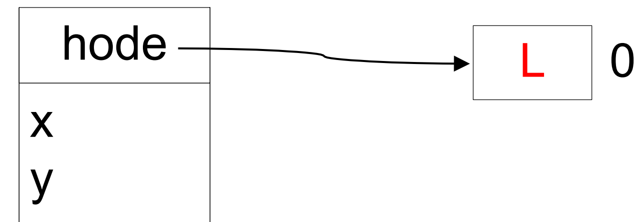
- Kompilatorsjekk av rX.f(...), både virtuelle og ikke-virtuelle): f må være definert i X eller i superklassen til X
- De ikke-virtuelle bindes ferdig i kompilatoren
- De virtuelle nummereres (med 'offset') fra første klasse med en virtuell – redefinisjoner får samme nummer
- La objekthodene inneholde en peker til klassens felles virtuell-tabell
-

rA.g(...) implementeres slik:

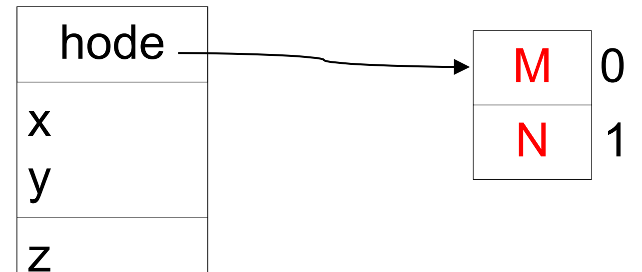
call(rA.virttab[g_offsett])

Kompilatoren vet: g_offset = 0
h_offset = 1

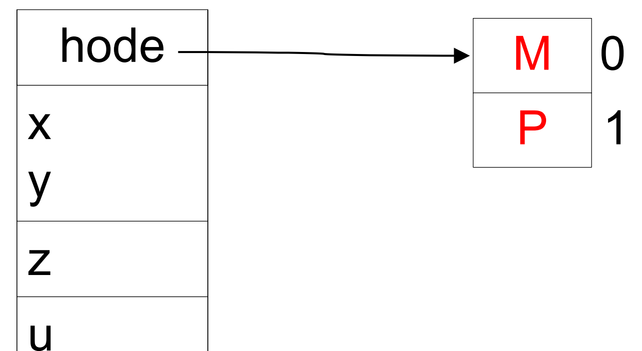
A-objekt



B-objekt

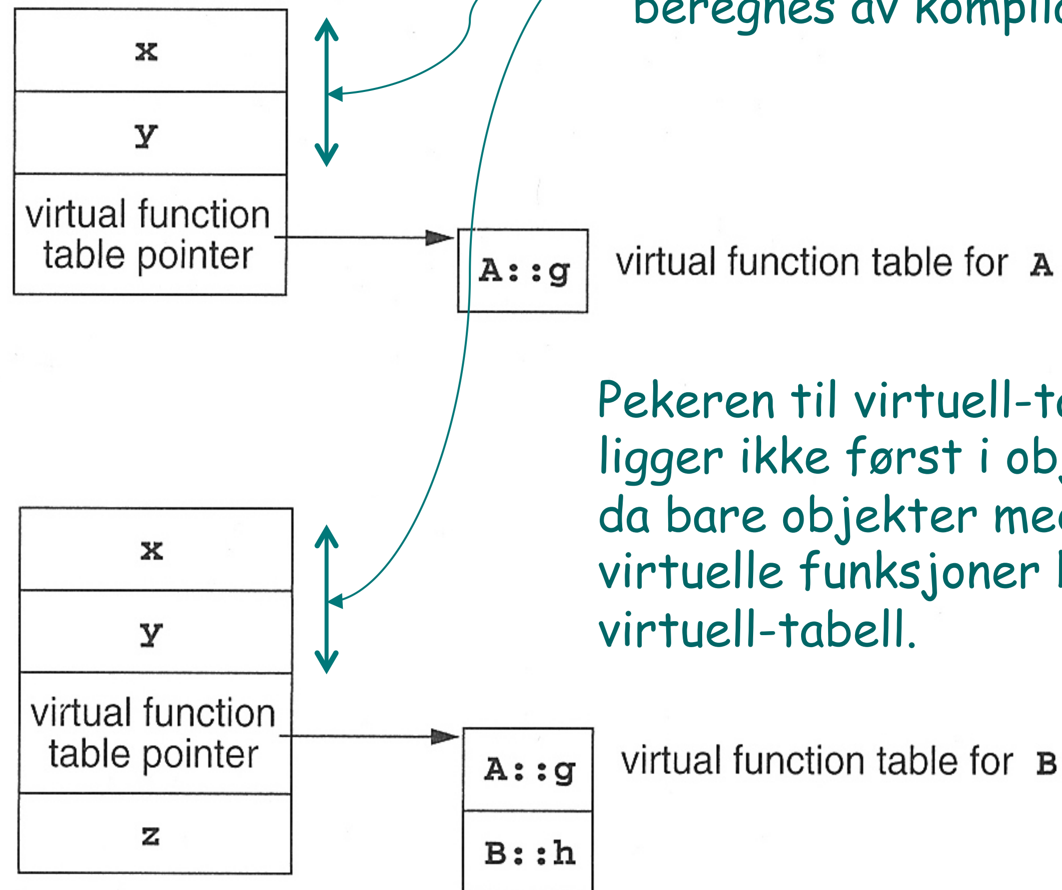


C-objekt



Impl. av virtuelle metoder som i boken (C++)

```
class A
{ public:
  double x,y;
  void f();
  virtual void g();
};

class B: public A
{ public:
  double z;
  void f();
  virtual void h();
};
```



Virtuell-tabell pekeren har relativ-adresse som kan beregnes av kompilatoren

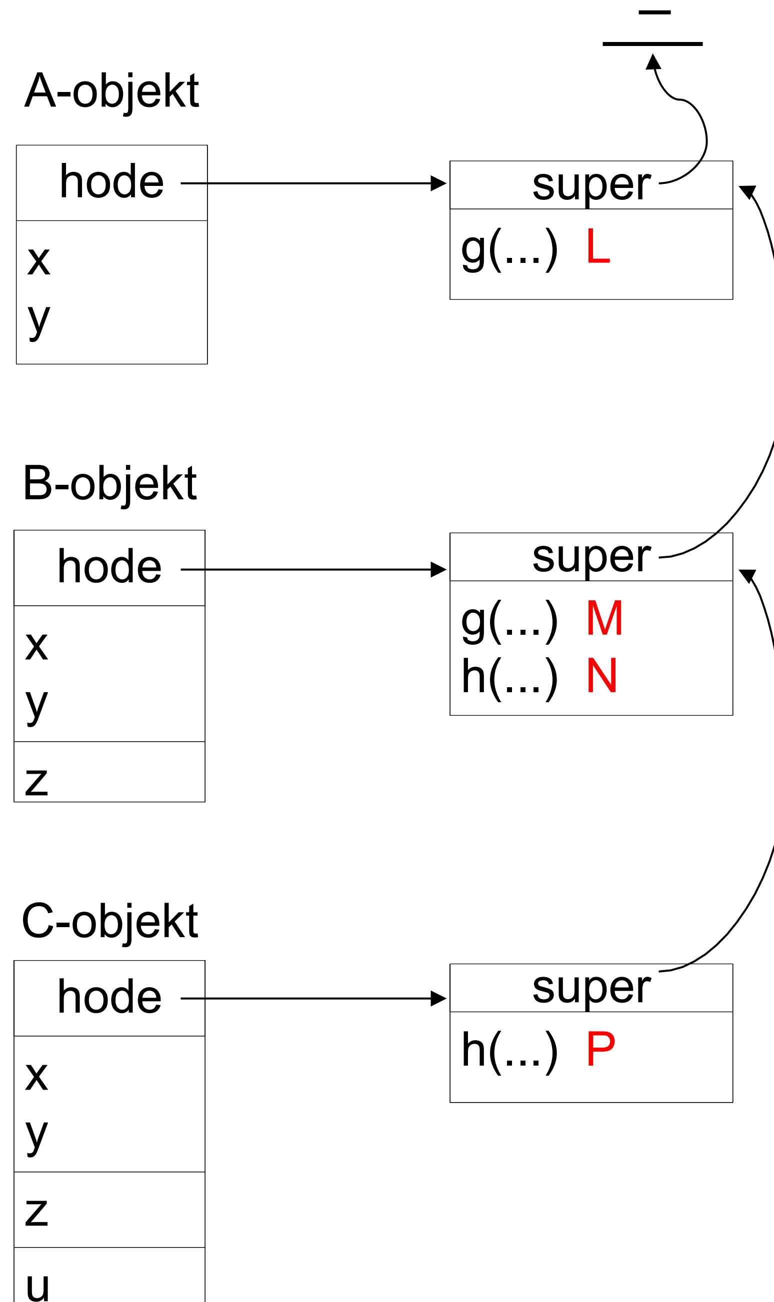
Pekeren til virtuell-tabellen ligger ikke først i objektet, da bare objekter med virtuelle funksjoner har en virtuell-tabell.

Utypede pekere (f.eks. Smalltalk)

- Ikke-virtuelle metoder finnes ikke
- Problem med virtuell-tabeller: Alle virtuell-tabeller måtte innholde alle metoder i alle klasser, altså for stor.
- I tillegg: I Smalltalk kan man legge til metoder underveis
- Derfor (antar at f er fjernet):

r.g(...) implementeres slik:

1. Gå til objektets klasse
2. Let etter 'g' ut gjennom superklassene



INF 5110 - 2014

PARAMETEROVERFØRING

’by value’ parameteroverføring (verdioverføring)

- Hver formell parameter blir implementert som en lokal variabel i prosedyren
- Ved kall blir disse variable satt slik:
  formell_var = aktuelt uttrykk
- I noen språk kan den formelle variabelen ikke forandres
- I C er ’by value’ eneste overføringsmåte. Man kan dog overføre pekere ’by value’

```
void inc2( int x)
/* incorrect! */
{ ++x;++x; }
```

```
void inc2( int* x)
/* now ok */
{ ++(*x);++(*x); }
```

Kall: inc2(&y)

```
void init(int x[],int size)
/* this works fine when called
   as init(a), where a is an array */
{ int i;
  for(i=0;i<size;++i) x[i]=0;
}
```

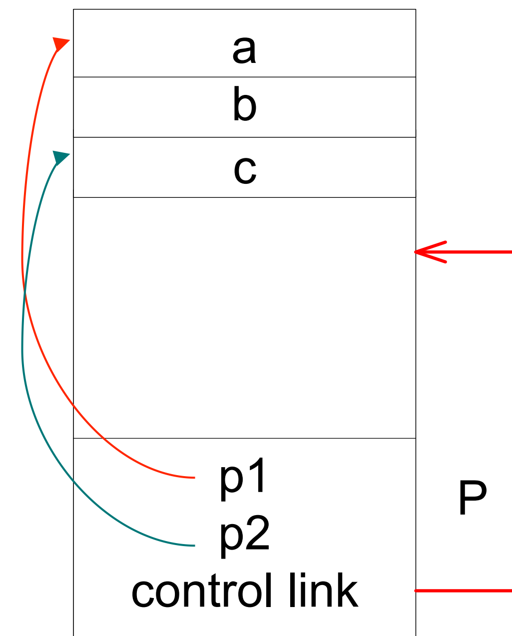
Kall: init(a)

INF 5110 - 2014

'by reference' parameteroverføring

- Overfører en peker/adresse til den aktuelle variabel
- Bare lov med variable som aktuelle parametere
- Fortran tillater imidlertid
  - P(5,b)
  - P(a+b,c)
- Passer til store datastrukturer

```
void P(p1,p2) {
  ...
  p1 = 3
}
var a,b,c;
P(a,c)
```




```
void inc2( int & x)
/* C++ reference parameter */
{ ++x;++x; }
```

Kall: inc2(y)

INF 5110 - 2014

’by value-result’ parameteroverføring

- Bare variable kan være aktuelle parametere
- Det allokeres en lokal variabel, som ved ’by value’
- Ved kallet gjøres
  formell_var = aktuell_var
- Ved exitutføres
  aktuell_var = formell_var
- Når bestemmes aktuell_var som skal brukes i aktuell_var = formell_var:
  - Ved kall
  - Ved exit
- ’by value-result’ kan gi effekt forskjellig fra ’by-reference’
  - Men ofte godkjennes dette som en implementasjon

```
void p(int x, int y)
{ ++x;
  ++y;
}

main()
{ int a = 1;
  p(a,a);
  return 0;
}
```

Eksempel på at det kan være forskjell på ‘by value-result’ og ‘by reference’

by value-result: 2

by reference: 3

’by name’ parameteroverføring

- Den aktuelle parameteren blir substituert inn for den formelle (’nesten’ rent tekstlig: den aktuelle parameteren beholder sitt skop, så altså ikke makro-ekspansjon)
- Om den aktuelle parameteren er et uttrykk blir det ikke beregnet før man bruker parameteren inne i prosedyren (lazy evaluering)
- Men: Uttrykket blir beregnet om igjen hver gang
- Implementasjon
  - Se den aktuelle parameteren (f.eks. et uttrykk) som en liten prosedyre (’thunk’)
  - Må optimaliseres for det tilfellet hvor parameteren er en enkel variabel (da er effekten som ved ’by reference)

```
void p(int x)
{ ++x; }
```

p(a[i])   ++a[i]

```
int i;
int a[10];

void p(int x)
{ ++i;
  ++x;
}

main()
( i = 1;
  a[1] = 1;
  a[2] = 2;
  p(a[i]);
  return 0;
}
```

Bedre eksempel

```
int i; int a[];

swap(int a, b) {
  int i;
  i = a;
  a = b;
  b = i;
};
...

i=3;
a[3]=6;

swap(i, a[i]);
```

$i_{lokal} = i_{global}$: $i_{lokal} = 3$
$i_{global} = a(i_{global})$: $i_{global} = a(3)$: $i_{global} = 6$
$a(i_{global}) = i_{lokal}$: $a(6) = 3$

INF 5110 - 2014

’by name’ eksempel

```
procedure P(par); name par; int par;
begin
  int x,y;
  ...
  par := x + y;        a)
  ...
  x := par + y;        b)
  ...
end;
...
P(v);
P(r.v);
P(5);
P(u+v);
```

	v	r.v	5	u + v
a)	OK	OK	feil	feil
b)	OK	OK	OK	OK

Neste forelesning onsdag 9. april

- Garbage collection
- Oppgaver
  - Stackorganisering: 7.2, 7.4, 7.10
  - Virtuelle metoder: 7.13,
  - Parameteroverføring: 7.15, 7.16

Eksamen 2005 a)

Anta ta vi har et språk med klasser og subklasser. Alle metoder er virtuelle, slik at de kan redefineres i subklasser.
Gitt følgende klassedefinisjoner:

```
class A { int i;
          void P {... AP ...};
          void Q {... AQ ...};   }
class B extends A { int j;
                    void Q {... BQ ...};
                    void R {... BR ...};   }
class C1 extends B {
  void P {... C1P ...};
  void S {... C1S ...};   }
class C2 extends B { int k;
                     void R {... C2R ...};
                     void T {... C2T ...};   }
```

Vis hvordan objekter av klassene C1 og C2 vil være strukturert (layout) og tegn virtuell-tabellen for hvert av objektene. Bruk navnene i metodekroppene til å angi hvilken definisjon som gjelder for hvert objekt.

Eksamen 2005

b)

Vi finner nå på å innføre i språket muligheten for å spesifisere en metode til å være **final**. Det skal bety at den ikke lenger er virtuell, dvs at den ikke kan redefineres i subklasser.

Anta at vi i klassen B spesifiserer metoden Q til å være **final**.

Må vi da endre på virtuell-tabellen for B-objekter?

Begrund svaret.

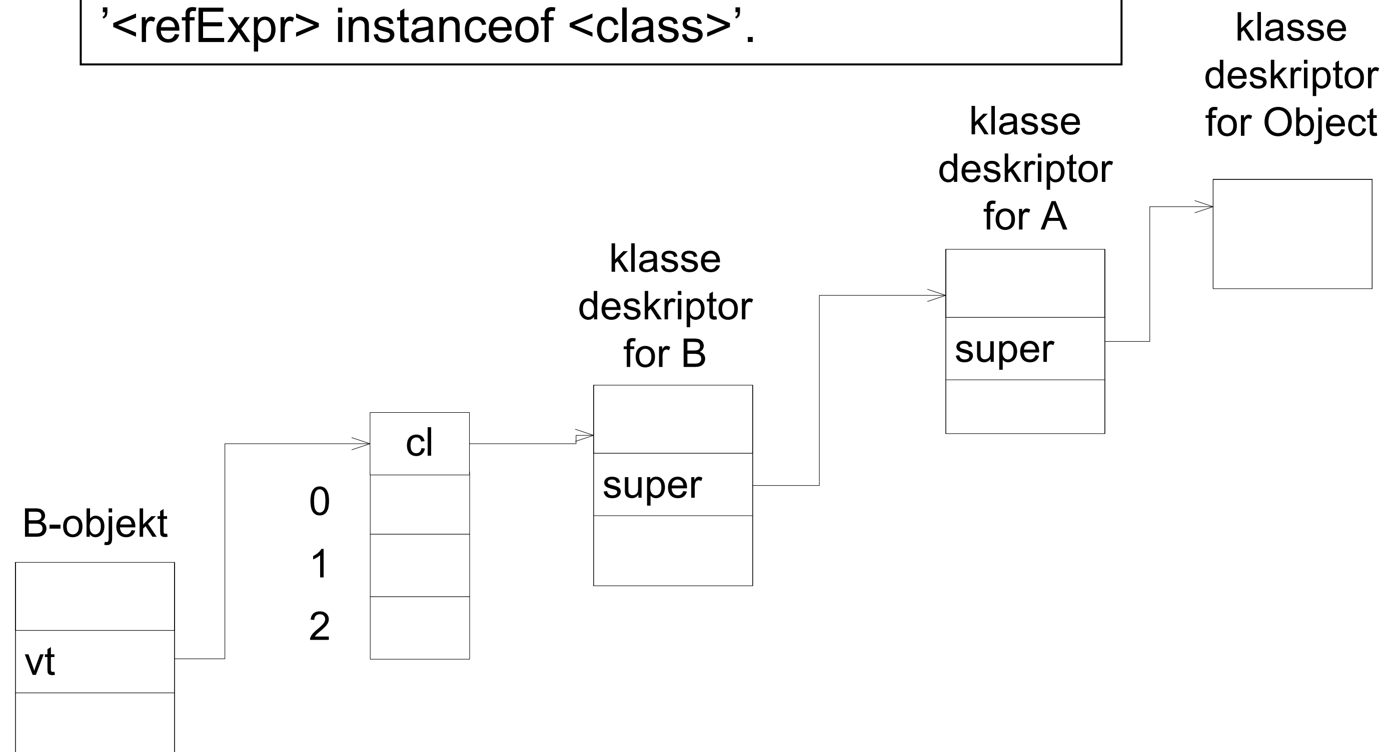
Eksamen 2005 c)

Vi innfører nå operatoren 'instanceof': Uttrykket

'<refExpr> instanceof <class>'

er True hvis objektet som <refExpr> peker på har en klasse som er klassen <class> eller en subklasse av klassen <class>, ellers False.

For å kunne implementere denne operatoren utvider vi virtuell-tabellen med en peker til klasse-deskriptoren, som det er en av for hver klasse i programmet. Klassedeskriptoren har en variabel 'super', som peker til klasse-deskriptoren for superklassen. Klasser uten eksplisitt superklasse har den spesielle klasse Object som super. Eksemplet viser dette for et objekt av klassen B.



Eksamen 2005 d)

For å effektivisere testen på 'instanceof' innfører vi at en klassedeskriptor har en tabell 'supers', som inneholder superklassene for klassen samt klassen selv. Denne tabellen har som indeks klassens 'subklassenivå' startende med 0 for Object, 1 for rotklassen i et subklassehierarki, 2 for neste nivå, osv. I vårt eksempel har klassen A subklassenivå 1, B har 2, C1 og C2 har begge 3.

Forklar hvordan denne tabellen kan effektivisere instanceof-testen.