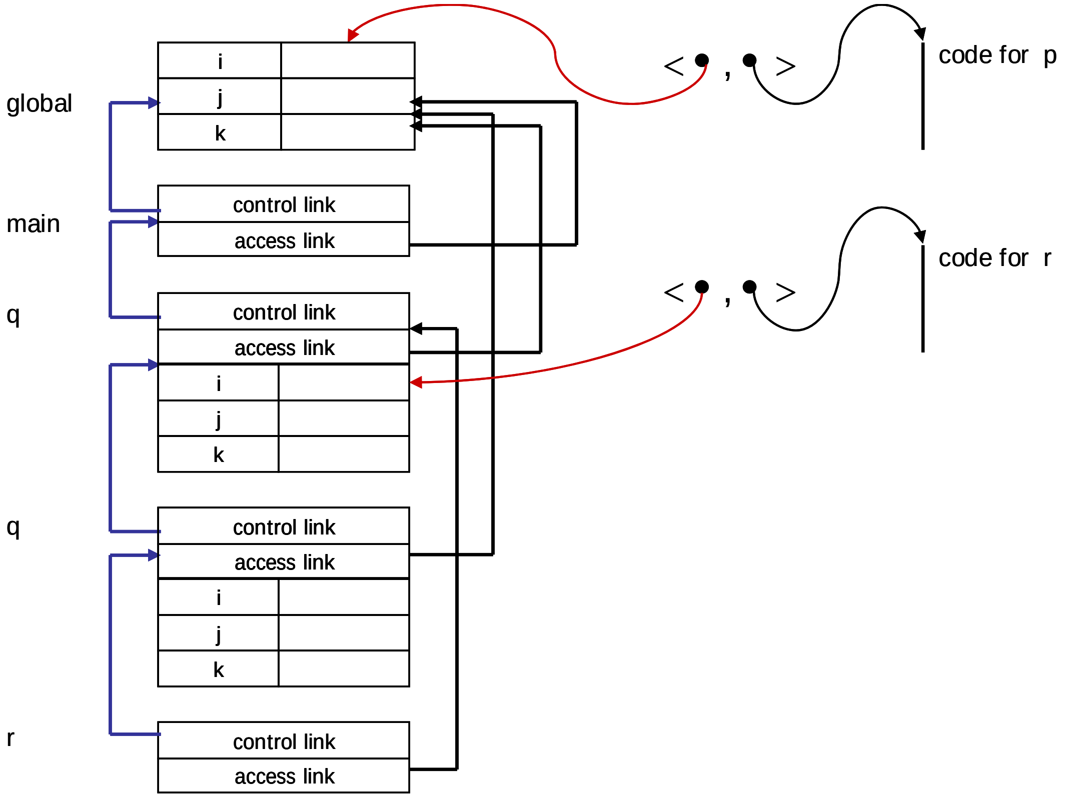
# Oppgave 1 Runtime-systemer, skoping, typer (vekt 40%)

## 1a

| | statisk scoping | dynamisk scoping |
|---|---|---|
| 1. utførelse av en print(k) | **3** | **9** |
| 2. utførelse av en print(k) | **6** | **9** |
| 3. utførelse av en print(k) | **9** | **9** |
| 4. utførelse av en print(k) | **6** | **9** |

## 1 b & c



## 1d

1)

If dist' < dist, then it should be possible to substitute a call dist(somePoint) with a call dist'(somePoint). dist' will be able to manipulate properties of ColorPoint, that an object of class Point will not have. We therefore have to check at run-time if somePoint points to a Point or ColorPoint object.

However, with dist < dist' a call dist'(someColorPoint) may be substituted with the call dist(someColorPoint), and as dist can only use properties defined for Point, and actual parameters must be of class ColorPoint they will always have Point properties.

## 1e

2)

If somePoint < somePoint', then it should be possible to substitute a call somePoint'() with a call somePoint(). Such a call may be:

```
 aColorPoint = somePoint'(1)
```

If  somePoint'(1) is substituted with somePoint(1), then one will try to assign a Point object to the reference aColorPoint which is typed with ColorPoint, and this will require a run-time type check.

However, if somePoint' < somePoint, then a call somePoint() may just as well be ssubstituted with a call somePoint'(). Such a call may be:

```
  aPoint = somePoint(1)
```

If somePoint(1) is substituted with somePoint'(1), then one will try to assign a ColorPoint object to the reference aPoint typed by Point, and this will always be legal.

# Oppgave 2 ML (vekt 40%)

## 2a

```
      fun f(g,h) = g(g(h)) * 3;

  0. w = int -> int -> int
  1. t = r -> s
  2. t = s -> b
  3. w = b -> c
  4. c = int -> d
  5. e = t * r -> d

  From 1 & 2:
  6. r = s = b

  From 3 & 4:
  7. w = b -> int -> d

  From 0 & 7:
  8. b = int
  9. d = int

  From 6 & 8:
  10. r = int
  11. s = int

  From 1, 10 & 11:
  12. t = int -> int

  From 9, 10 & 12:
  13. e = (int -> int) * int -> int
```

## 2b

```
  1) fun firstElems xs = map hd xs;
  2) fun prodFirstElems xs = foldl op* 1 (firstElems xs);
       Another solution:
       fun prodFirstElems2 xs = foldr op* 1 (firstElems xs);
```

## 2c

ONE SOLUTION:

```
  fun f(0,count) = count
       |  f(1,count) = 0.0/0.0
       |  f(3,count) = ~1.0
       |  f(x,count) = f(x-2, count+1.0);
```

ANOTHER POSSIBLE SOLUTION:

```
  fun fAux(0,count, flag) = count
       |  fAux(1,count, flag) = if flag=1 then 0.0/0.0 else ~1.0
       |  fAux(x,count, flag) = fAux(x-2, count+1.0, flag)

  fun eqTof(1,count) = fAux(1, count, 1)
    | eqTof(x,count) = fAux(x, count, 0);
```

# Oppgave 3 Prolog (vekt 20%)

## 3a

a)

For example:

```
mother(sarah,per).
mother(sarah,anne).
mother(anne,sofia).
mother(sofia,carlos).
```

b)

```
son(X,Y) :- father(Y,X), male(X).

daughter(X,Y) :- father(Y,X), female(X).

brother(X,Y) :- father(Z,X), father(Z,Y), X\==Y.

parent(X,Y) :- father(x,Y).
parent(X,Y) :- mother(x,Y).

uncle(Uncle,Person) :- brother(Uncle,Parent), parent(Parent,Person).
```

c)

- All the persons who have at least one brother.

Define first:

```
atleastonebrother(X) :- (son(X,Y); daughter(X,Y)), son(Z,Y), X\==Z.
  % Alternative: atleastonebrother(X) :- brother(X,Y), male(Y).


atleastonebrother(X).
```

- All the persons who are uncle of a female

Define first:

```
unclefemale(X) :- uncle(X,Y), female(Y).

unclefemale(X).
```

## 3b

Natural numbers may be defined as follows in Prolog:

```
natural_number(0).
natural_number(s(X)) :- natural_number(X).


plus(0,X,X) :- natural_number(X).
plus(s(X),Y,s(Z)) :- plus(X,Y,Z).
```

```
prod(0,X,0) :- natural_number(0).
prod(s(X),Y,Z) :- prod(X,Y,XY), plus(XY,Y,Z).

exp(s(N),0,0).
exp(0,s(X),s(0)).
exp(s(N),X,Y) :- exp(N,X,Z), prod(Z,X,Y).
```