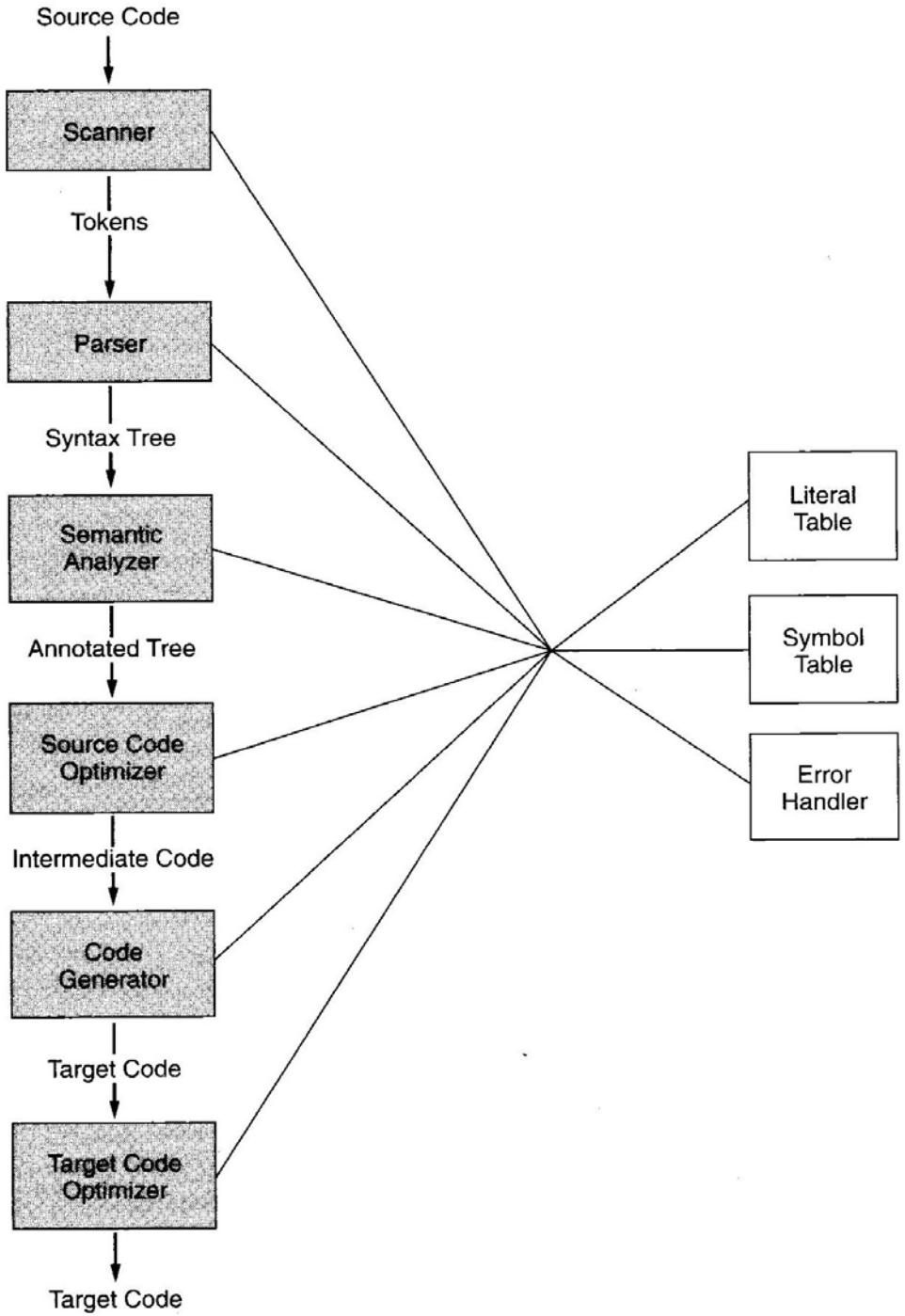
Runtime-systemer del I

Kapittel 7



INF 5110 - 2013

Oversikt

- Generell lagerorganisering (kap 7.1)
- Språk som bare trenger statisk allokering (kap 7.2)
- Språk som trenger stakk-orientert allokering (kap 7.3)
- Språk som trenger mer generell allokering (kap 7.4)
- Parameteroverføring (kap 7.5)

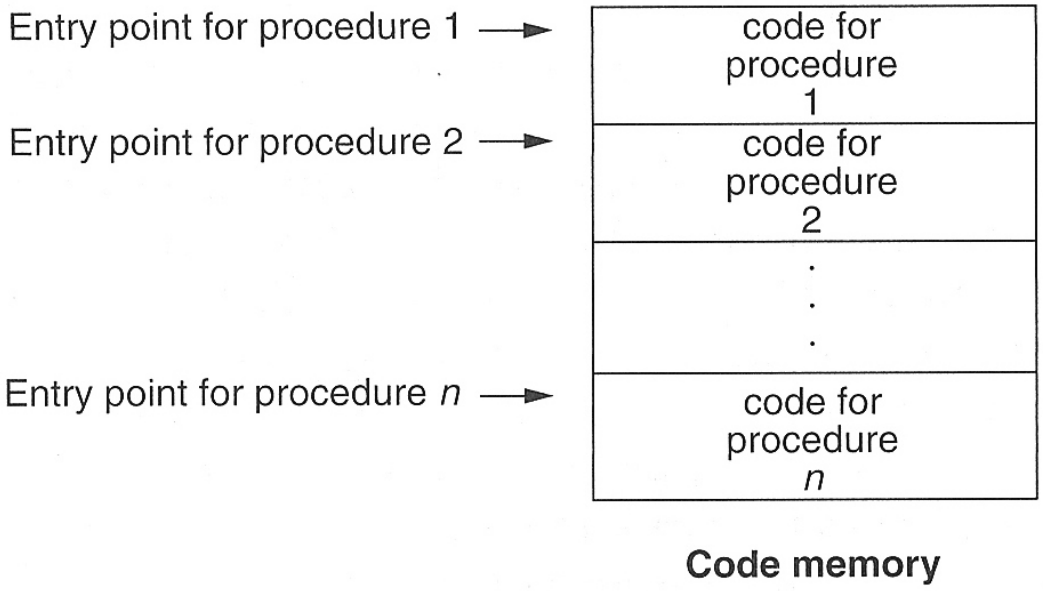
Avhenger av begrepene i språket

INF 5110 - 2013

Den oversatte programkoden

- kan nesten altid betraktes som statisk allokert
- skal hverken flyttes eller forandres under utførelse

- Kompilatoren kjenner alle adresser til kodebiter

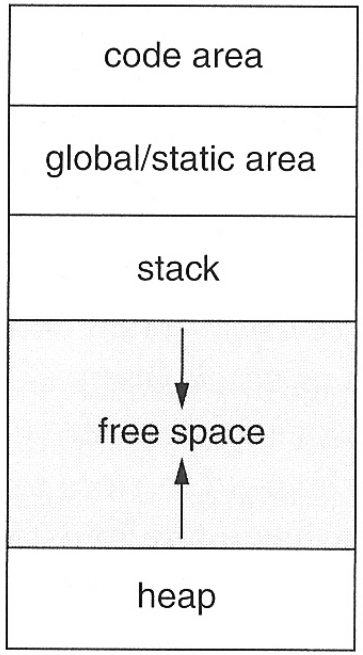


Men husk

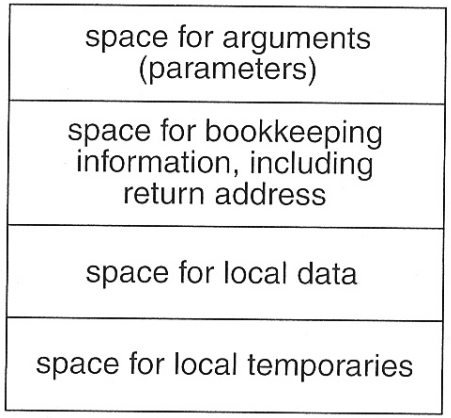
Koden blir ofte produsert som relokerbare kode, som får sin endelige plassering av linker/loader

Lagerorganisering

- Typisk organisering under utførelse dersom et programmeringsspråk har alle slags data (statisk, stakk, dynamisk)



- Typisk organisering av data for et prosedyrekall (aktiveringsblokk)

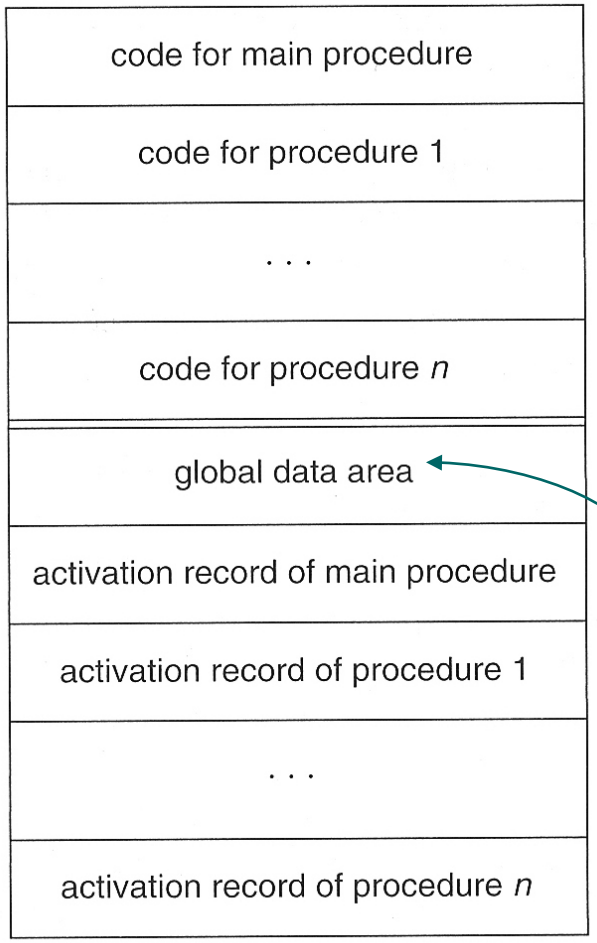


Det er gjerne ut fra plasseringen her man karakteriserer språk til være
- statisk organisert
- stakk-organisert
- heap/dynamisk organisert

Kapittel 7.2

STATISK ORGANISERING

Full statisk organisering (eks. Fortran)

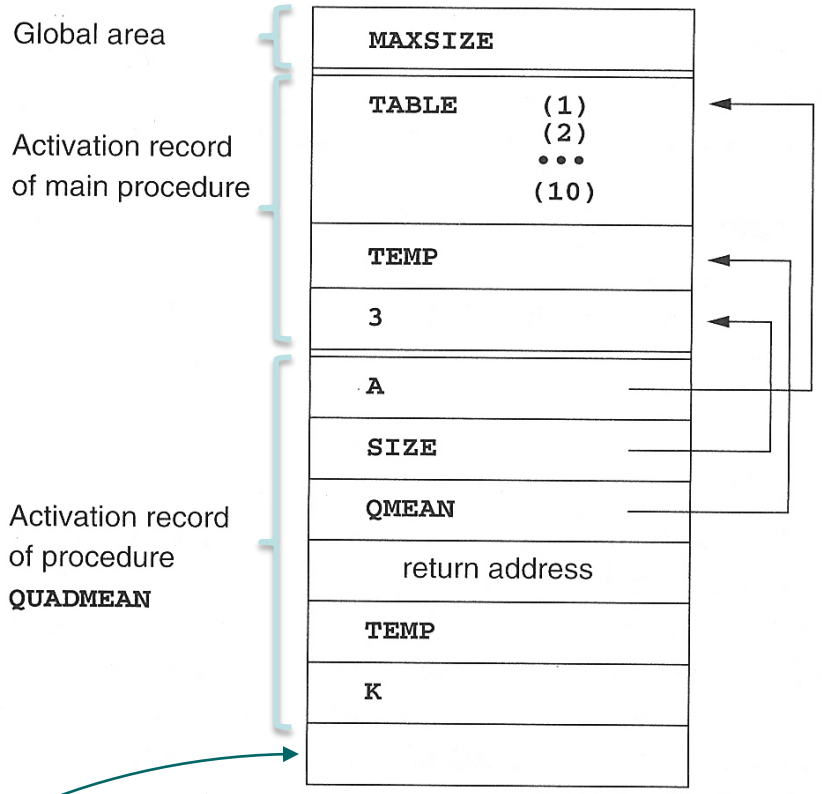


- Kompilatoen kan beregne hvor alt ligger
  - Utførbar kode
  - Variable
  - Alle slags hjelpedata

bl.a. alle slags større konstanter i programmet

Et eksempel i Fortran

```
      PROGRAM TEST
      COMMON MAXSIZE
      INTEGER MAXSIZE
      REAL TABLE(10),TEMP
      MAXSIZE = 10
      READ *, TABLE(1),TABLE(2),TABLE(3)
      CALL QUADMEAN(TABLE,3,TEMP)
      PRINT *, TEMP
      END

      SUBROUTINE QUADMEAN(A,SIZE,QMEAN)
      COMMON MAXSIZE
      INTEGER MAXSIZE,SIZE
      REAL A(SIZE),QMEAN, TEMP
      INTEGER K
      TEMP = 0.0
      IF ((SIZE.GT.MAXSIZE).OR.(SIZE.LT.1)) GOTO 99
      DO 10 K = 1,SIZE
         TEMP = TEMP + A(K)*A(K)
10    CONTINUE
99    QMEAN = SQRT(TEMP/SIZE)
      RETURN
      END
```



Plass til mellomresultater o.l. Kompilatoren kan beregne hvor mye som trengs

I Fortran overføres parametere som pekere til de aktuelle verdier/variable

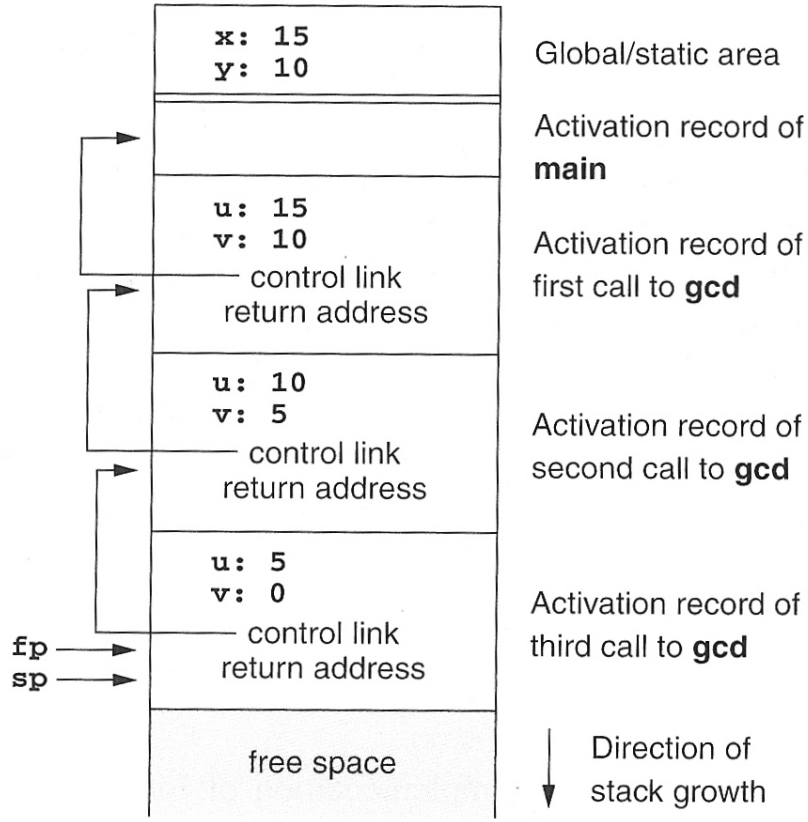
INF 5110 - 2013

Kapittel 7.3

STAKK-ORGANISERING

Et eksempel i C

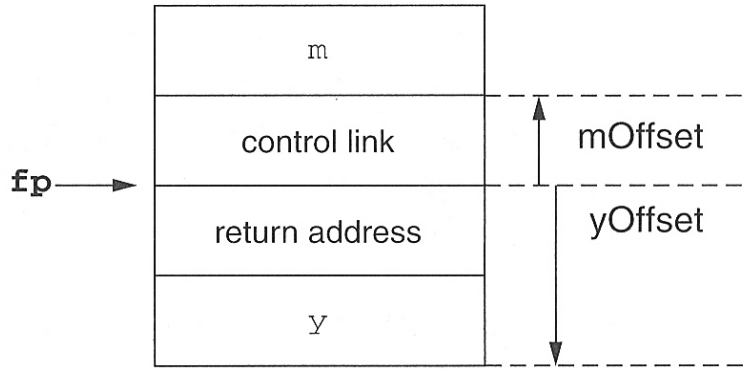
```
#include <stdio.h>

int x,y;

int gcd( int u, int v)
{ if (v == 0) return u;
  else return gcd(v,u % v);
}

main()
{ scanf("%d%d",&x,&y);
  printf("%d\n",gcd(x,y));
  return 0;
}
```



Variabel-aksess

```
void g(int m)
{ int y = m-1;
  if (y > 0)
  { f(y);
    x--;
    g(y);
  }
}
```

Layout av g sin aktiveringsblokk:



Hvordan utføre et kall

```
void g(int m)
{ int y = m-1;
  if (y > 0)
  { f(y);
    x--;
    g(y);
  }
}
```

Ved prosedyre-kall (entry):

1) Beregn parameter-verdiene og push dem på stakken.
2) control link := frame pointer
3) frame pointer := stack pointer
4) Lagre returadressen (hvis nødvendig).
5) Hopp til maskinkoden for aktuell prosedyre.
6) Sett av plass til lokale variable ved å flytte sp.

Ved prosedyre-slutt (exit):

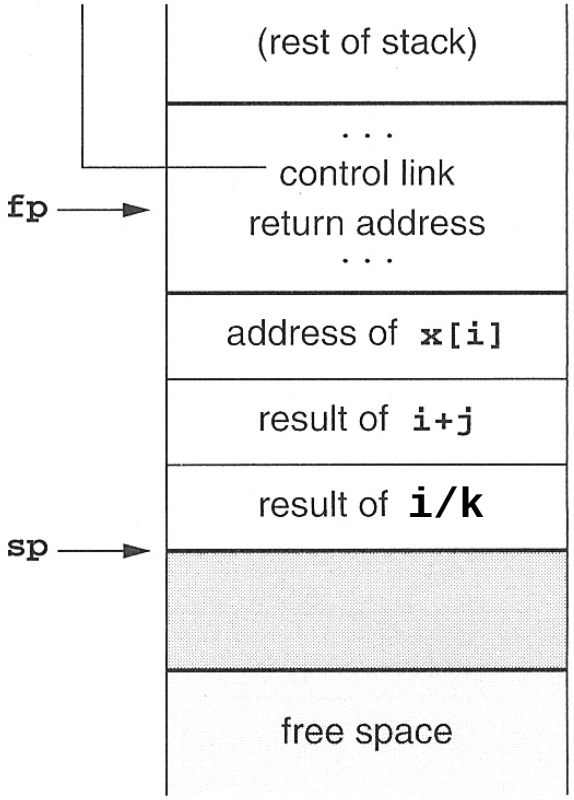
1) stack pointer := frame pointer
2) frame pointer := control link
3) Hopp til returadressen.
4) Fjern parametrene fra stakken (ved å flytte sp).

Behandling av mellomresultater

```
x[i] = (i + j)*(i/k + f(j))
```

adresse verdi verdi

Antar strikt beregning fra venstre mot høyre. Kallet f(j) kan forandre verdier.



Activation record of procedure containing the expression

Stack of temporaries

New activation record of call to `f` (about to be created)

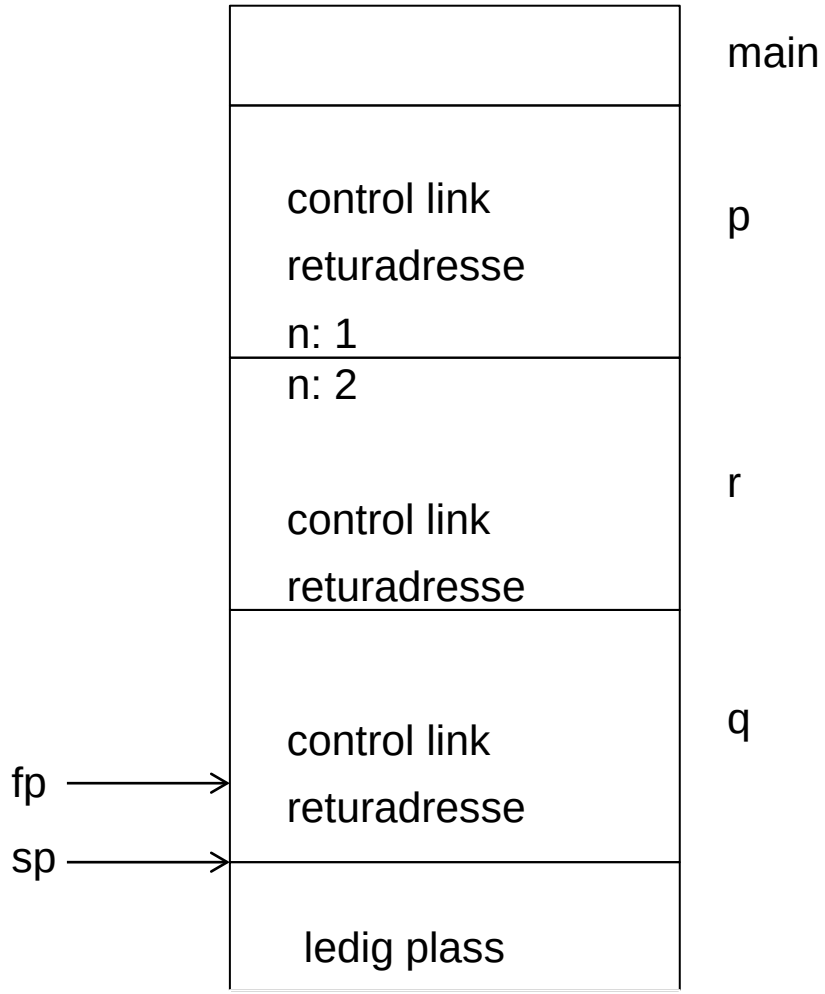
Trenger ikke sette av fast maksimal plass til slike mellomresultater for hele blokkens levetid. I motsetning til hva man naturlig gjør i Fortran.

OPPGAVE 7.1 + 7.2

Prosedyrer inne i prosedyrer

Et første forsøk

```
program nonLocalRef;

procedure p;
var n: integer;

      procedure q;
      begin
        (* a reference to n is now
          non-local non-global *)
      end; (* q *)

      procedure r(n: integer);
      begin
        q;
      end; (* r *)

begin (* p *)
  n := 1;
  r(2);
end; (* p *)

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

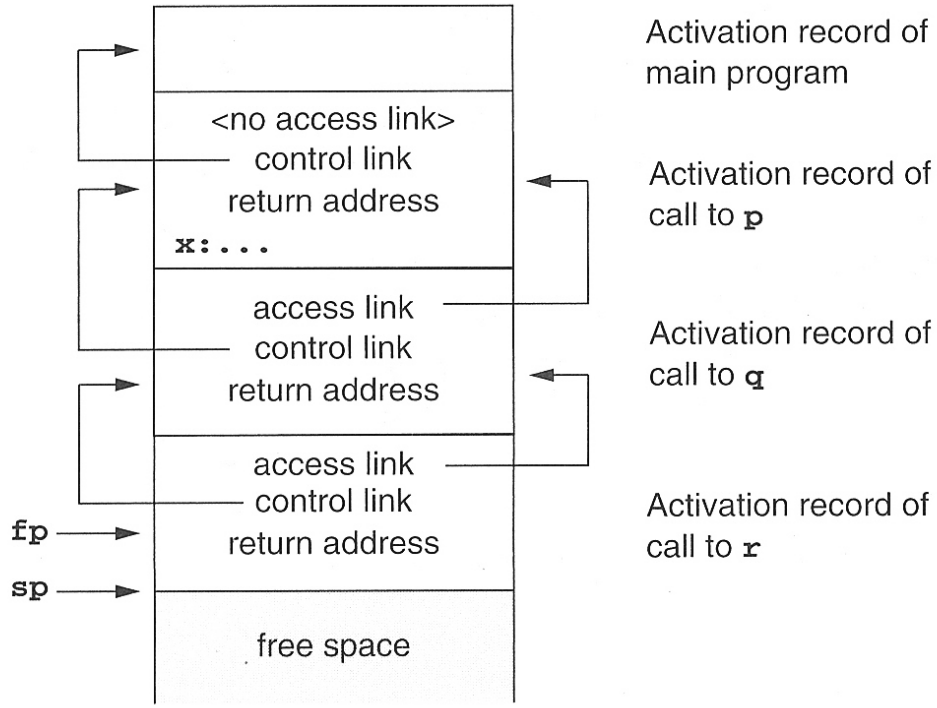


Hvordan kan vi aksessere 'n' i 'p' ?

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



fp.al.al.x

diff i blokknivå

Hvordan utføre et kall – med nestede nivåer

Ved prosedyre-kall (entry):

1) Beregn parameter-verdiene og push dem på stakken.
2) Følge access link fra fp like mange ganger som forskjellen i blokk-nivå mellom den kalte og kallstedet (0 hvis lokal), og push den på stakken.
3) control link := frame pointer
4) frame pointer := stack pointer
5) Lagre returadressen (hvis nødvendig).
6) Hopp til maskinkoden for aktuell prosedyre.
7) Sett av plass til lokale variable ved å flytte sp.

Ved prosedyre-slutt (exit):

1) stack pointer := frame pointer
2) frame pointer := control link
3) Hopp til returadressen.
4) Fjern parametrene + access link fra stakken (ved å flytte sp).

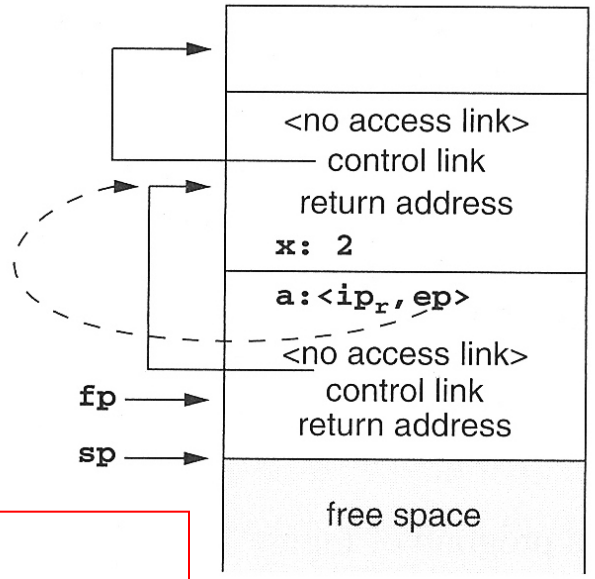
Prosedyrer som parametere

```
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.
```

Dette må da oversettes helt spesielt:

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

ip_r
ep_r



Activation record of main program

Activation record of call to `q`

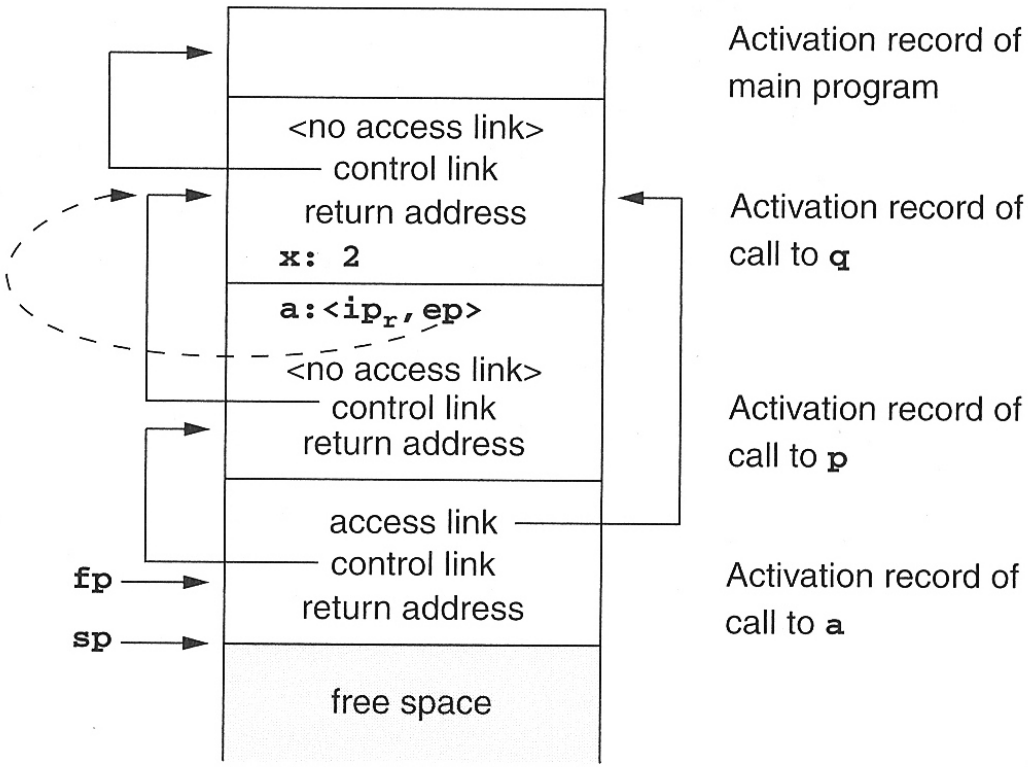
Activation record of call to `p`

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:



OPPGAVE 7.4, 7.5 + 7.10

Neste forelesning: Tirsdag 9. april (Sed)

Runtimesystemer del II