Dagens tema

- Kjøresystemer
  (Ghezzi&Jazayeri 2.6, 2.7)
  — Bokholderi og minneorganisering
  — Forskjellige språkklasser

Språk med rekursive rutiner

Språket C3 er C2 utvidet med

- muligheten til å kalle rutiner rekursivt.

Eksempler: C, PASCAL.

Problem
Hver rutines aktiveringspost kan forekomme 0 eller flere ganger.

Løsning
Legg aktiveringspostene på stakken. LIFO-disiplin!

Hver aktiveringspost må inneholde en peker til forrige aktiveringspost. Denne kalles *dynamisk link*.

Implementasjon

Ved implementasjon må vi også ha:

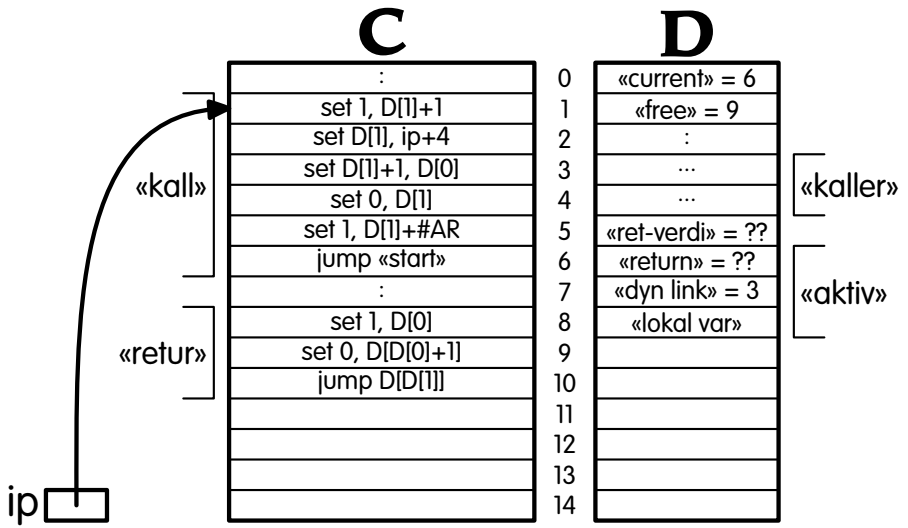
D[0] inneholder en peker current til nåværende aktiveringspost, og

D[1] har en peker free til første frie lokasjon på stakken.

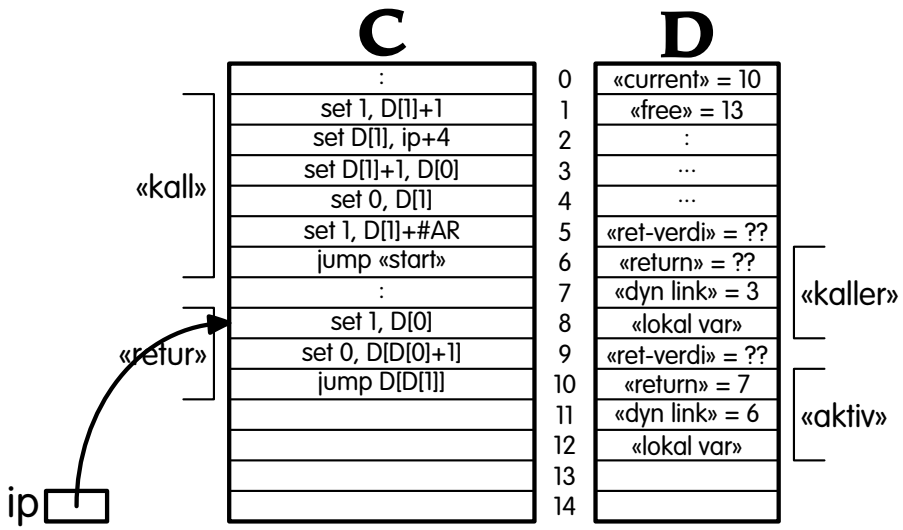
Lokale variable kan nå aksesseres som

D[0]+*tillegg*

Før et kall er situasjonen slik:



Etter kallet ser det slik ut:



Aksess av variable

Lokale variable kan nå aksesseres ved å følge "current" til den aktive aktiveringsblokken og så legge til relativ addresse ("offset"). Adressen er

current+tillegg som er D[0]+*tillegg*

og verdien er da i

D[D[0]+*tillegg*]

Globale variable ligger et fast sted og kan aksesseres direkte:

D[*adresse*]

Dynamisk link

Hver aktiveringspost inneholder *dynamisk link* som er en peker til forrige aktiveringspost.

Kall og retur - mer generelt

```
1  current  == D[0]
2  free     == D[1]
3  RP       == D[current]
4  DL       == D[current+1]
```

Kall

```
1  free     += 1          -- sett av plass til retur-verdi
2  D[free]  = ip + 4      -- lagre RP
3  D[free+1] = current    -- lagre DL
4  current  = free        -- sett DL
5  free     += < 2+antall lokale variable>
6  ip       = < start av rutine>    -- sett i gang rutine
```

Retur

```
1  free = current         -- slett aktiv aktiveringsblokk
2  current = DL           -- la forrige aktiveringsblokk bli aktiv
3  ip   = D[free]         -- hopp til RP
```

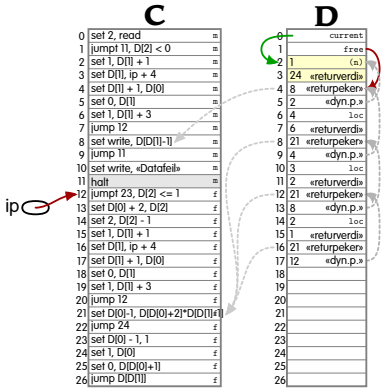

Et eksempel

En rekursiv fakultetsfunksjon kan skrives slik. Merk: vi har fortsatt ikke parametre.

```
1  int n;
2
3  int fact(){
4    int loc;
5    if (n > 1) {
6      loc = n--;
7      return loc * fact();
8    } else {
9      return 1;
10   }
11 }
12
13 main(){
14   scanf("%d", &n);
15   if (n >= 0)
16     printf("%d", fact());
17   else
18     printf("Datafeil!");
19 }
```

fact-eksempel - kjøring

SIMPLESEM-kode for fact, samt kjøring, ligger på en egen PDF-fil. Slik ser situasjonen ut etter kjøring:



Språk med blokker

Indre lokale deklarasjoner

I språket C4′ får vi lov å ha lokale variable i en sammensatt setning:

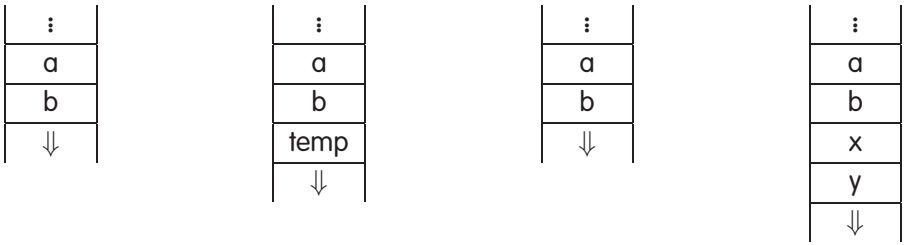
```
1  void f()
2  {
3    int a, b;
4      :
5    if (a < b) {
6      int temp = a;
7      a = b;  b = temp;
8    }
9      :
10   while (a > b) {
11     int x, y;
12       :
13   }
14 }
```

Alle språk med blokker (se ark 13) har denne muligheten, men også C.

Implementasjon

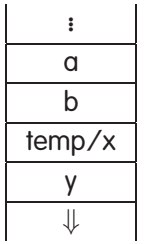
Dette krever minimale utvidelser i forhold til C3. Det er to måter å gjøre det på:

1 Utvide stakken ved hver ny deklarasjon (omtrent som ved rutinekall):



Implementasjon

2 Sette av plass allerede når rutinen kalles:



Det er mulig å spare plass ved å la variable dele lokasjoner.

Rutiner inni rutiner

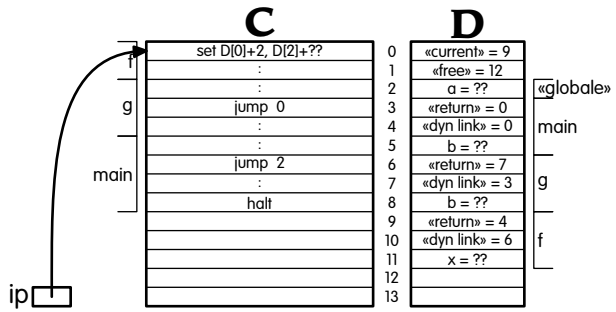
Eksempel

```
1  int a;
2
3  void main()
4  {
5    int b;
6    void f()
7    {
8      int x;
9      x = a + b;
10   }
11   void g()
12   {
13     int b;
14     f();
15   }
16   g();
17 }
```

C4" er et fullt blokkorientert språk hvor alle deklarasjoner kan plasseres inne i lokale blokker. Andre eksempler er Algol-60 og Simula.

Implementasjon

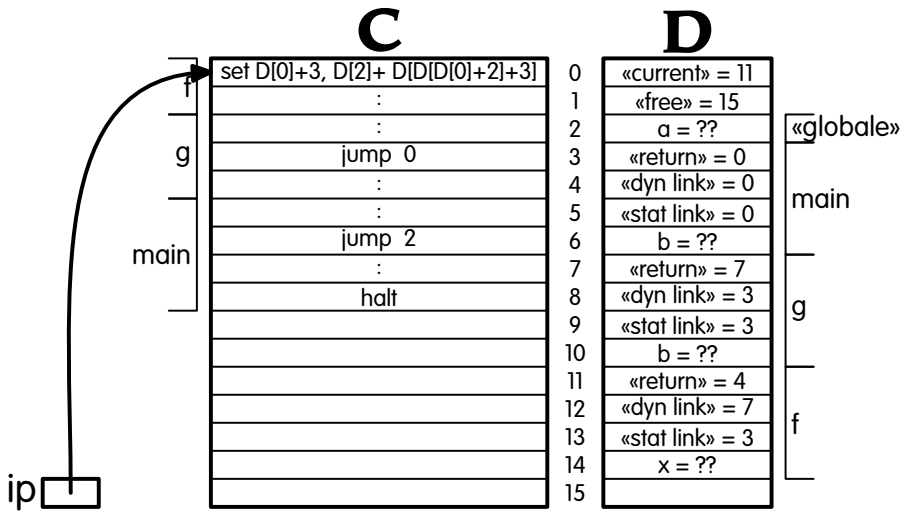
Hvis vi oversetter dette som vi gjorde med C3-språkene, får vi følgende kode:



Problem
Hvordan får vi tak i b i main?

Løsning
Vi trenger en *statisk link* som viser aktiveringsposten for omkringliggende blokk.

Med statisk link ser bildet slik ut:



Variable aksesseres slik:

- Globale variable ligger et fast sted.
- Lokale variable aksesseres via current.
- Variable på mellomnivåene får vi tak i ved å følge statisk link bakover et visst antall ganger. Dette antallet kan avgjøres under kompileringen.

Verdien til en ikke-lokal variabel kan skrives:

$$D[fp(d) + tillegg]$$

der $tillegg$ er relativ adresse og
d er antall ganger SL skal følges, og
der fp-funksjonen "frame pointer") er definert ved:

$$fp(d) == \texttt{ if } d = 0 \texttt{ then } D[0] \texttt{ else } D[fp(d-1) + 2]$$

Siden $fp(0) = current$ kan denne strategien brukes for lokale variable også!

For eksempel har vi at

$$fp(1) = D[current + 2]$$

som svarer til å følge SL én gang, og

$$fp(2) = D[D[current + 2] + 2]$$

som svarer til å følge SL to ganger.