



Dagens tema

• Kjøresystemer

(Ghezzi&Jazayeri 2.6, 2.7)

- Innledende om semantikk
- Operasjonell semantikk / SIMPLESEM
- Bokholderi og minneorganisering
- Forskjellige språkklasser

1/23

Forelesning 9 – 22.10.2003

Semantikk

Hva kan du *gjøre* med språket ditt?

Forelesning 9 – 22.10.2003

2/23

Semantikk

Hva kan du *gjøre* med språket ditt?

- En måte å svare på: gi **semantikken** til språket!
 ... en beskrivelse av *hva som skjer* når programmer utføres.

Forelesning 9 – 22.10.2003

2/23

Semantikk

Hva kan du *gjøre* med språket ditt?

- En måte å svare på: gi **semantikken** til språket!
 ... en beskrivelse av *hva som skjer* når programmer utføres.

Dette er langt fra trivielt!

Forelesning 9 – 22.10.2003

2/23

Semantikk

Hva kan du *gjøre* med språket ditt?

- En måte å svare på: gi **semantikken** til språket!
... en beskrivelse av *hva som skjer* når programmer utføres.

Dette er langt fra trivielt!

For å beskrive syntaks har vi noen standard verktøyer. (BNF, syntaksdiagrammer, etc.)
Tilsvarende orden har vi ikke for semantikk.

Semantikk

To innfallsvinkler:

- denotasjonell semantikk
 - beskrive i forhold til en matematisk modell
 - ofte svært komplisert
- operasjonell semantikk
 - **SIMPLESEM** - en abstrakt maskin
 - gi en semantikk ved å oversette inn i SIMPLESEM

Semantikk

To innfallsvinkler:

- denotasjonell semantikk
 - beskrive i forhold til en matematisk modell
 - ofte svært komplisert
- operasjonell semantikk
 - **SIMPLESEM** - en abstrakt maskin
 - gi en semantikk ved å oversette inn i SIMPLESEM

Vårt fokus her vil være på **minneorganisering** og **bokholderi**.
Krav til minneorganisering sier noe om hvor *rikt* et språk er.

Bokholderi

Det viktigste bokholderiet et kjøresystem må ta seg av, er følgende:

- Variable
 - Hvordan settes det av plass?
 - Hvordan fjernes de igjen?
 - Hvordan finner man variablene?
- Rutiner
 - Hvordan kaller man en rutine?
 - Hvordan kommer man tilbake fra en rutine?
 - Hvordan overfører man parametre?
- Blokker
 - Hvordan går man inn og ut av blokker?

Forskjellige språkklasser

Statiske språk – C1, C2, C2'

- minnekravet kan regnes ut før programmet kjøres, derfor kan minnet tildeles/allokeres *før* programstart
- ikke rekursjon

(FORTRAN, COBOL)

Stakk-baserte språk – C3, C4

- minnebruk følger LIFO-/stakk-prinsipp og kan *ikke* regnes ut ved kompilering
- minnet tildeles *automatisk* ved kjøring når variable deklarerer i et skop
- minnet frigjøres når en er ute av skopet

(ALGOL 60)

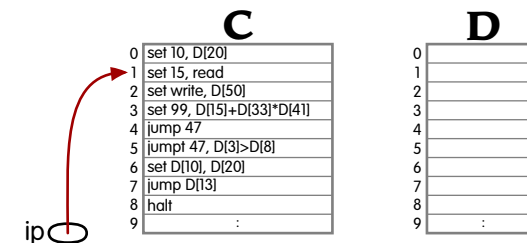
Dynamiske språk – C5

- minnebruk mer uforutsigbart, tildeles *dynamisk* ved behov
- minnet kan ikke organiseres i en stakk
- heap

(SIMULA)

SIMPLESEM

Maskinen **SIMPLESEM** er meget enkel:



C er et kodelager, adressert fra 0.

ip (instruksjonspeker) peker på neste instruksjon i **C**.

D er et datalager, adressert fra 0.

SIMPLESEM**Instruksjonsløkken**

For hver instruksjon gjør **SIMPLESEM** følgende:

1. Hent neste instruksjon: C[ip].
2. Øk ip: ip = ip+1.
3. Utfør instruksjonen.

SIMPLESEM**Instruksjonene**

set plasserer verdier i D:

```

1 set 10,D[20]
2 set 15,read
3 set write,D[50]
4 set 99,D[15]+D[33]*D[41]
5 set D[10],D[20]
```

jump hopper ved å sette ip.

```

1 jump 47
2 jump D[13]
```

jumpt hopper hvis en betingelse er sann («true»).

```
1 jumpt 47,D[3]>D[8]
```

halt avslutter programmet.

Statiske språk uten rutiner

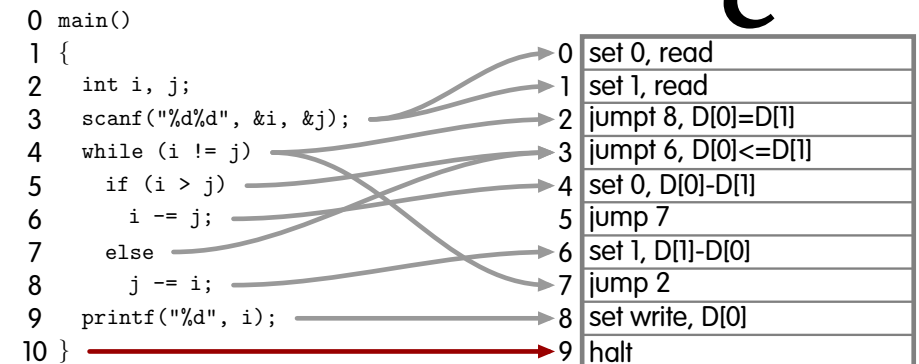
Språket **C1** er en særdeles forenklet form for C:

- Det finnes ingen rutiner eller blokker.
- Alle variable er følgelig globale.

Eksempel: CSH.


Variable

Alle variable ligger i D under hele kjøringen.

C1-eksempel - største felles multiplum

C1-eksempel - største felles multiplum

C		D	
0	set 0, read	0	4 (i)
1	set 1, read	1	4 (j)
2	jumpf 8, D[0]=D[1]	2	
3	jumpf 6, D[0]<=D[1]	3	
4	set 0, D[0]-D[1]	4	
5	jump 7	5	
6	set 1, D[1]-D[0]	6	
7	jump 2	7	
8	set write, D[0]	8	
9	halt	9	

ip 

Statiske språk med rutiner

C2 er C1 utvidet med

- rutiner med lokale variable, men uten parametre.

(Rekursive kall er ikke tillatt.)

Tildelingen av minne kan gjøres før kjøring. (Statisk allokering.)

For hver variabel kan det tilordnes en adresse i D-lageret. Denne adressen kan beholdes under hele kjøringen.

C2 i SIMPLESEM

Aktiveringspost

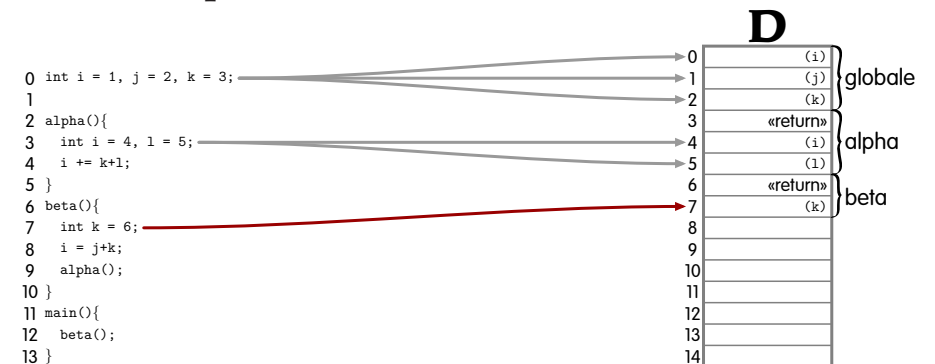
Hver rutine må ha en *aktiveringspost* i D. Der ligger:

- returadressen og
- lokale variable.

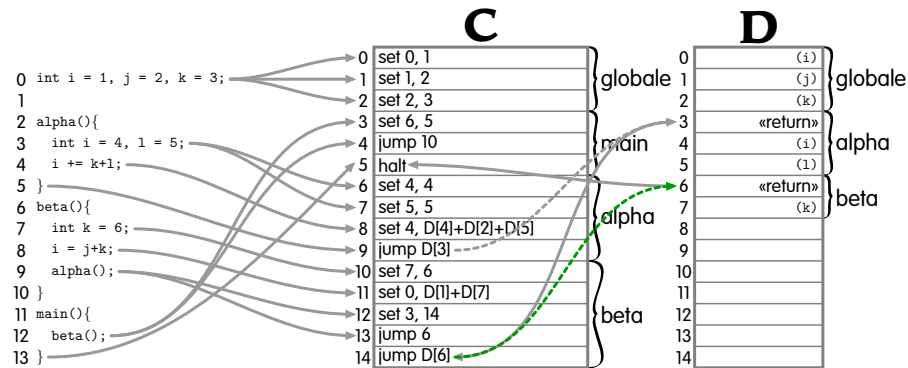
Dessuten må det av og til settes av plass til en returverdi.

C2-eksempel - oversettelse

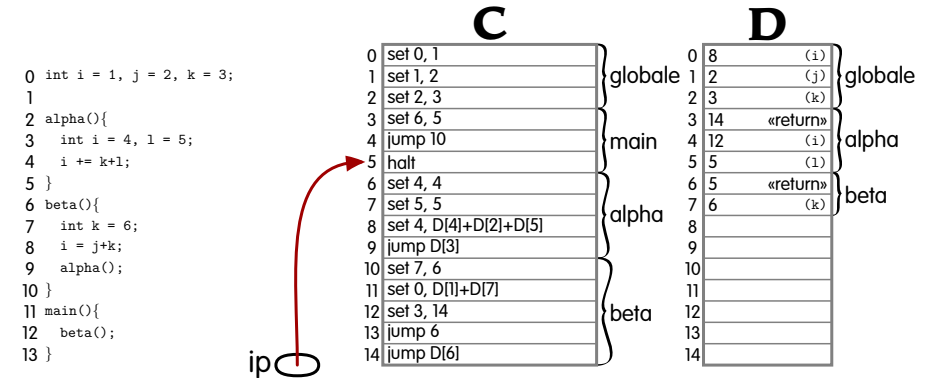
	D	
0 int i = 1, j = 2, k = 3;	0 (i)	globale
1	1 (j)	
2 alpha(){	2 (k)	
3 int i = 4, l = 5;	3 «return»	alpha
4 i += k+1;	4 (i)	
5 }	5 (l)	beta
6 beta(){	6 «return»	
7 int k = 6;	7 (k)	
8 i = j+k;	8	
9 alpha();	9	
10 }	10	
11 main(){	11	
12 beta();	12	
13 }	13	
	14	



C2-eksempel - oversettelse



C2-eksempel - kjøring



Språk med separat kompilering

Språket **C2'** er C2 med muligheter for separat kompilering:

Fil 1:

```

1 int i = 1, j = 2, k = 3;
2 extern beta();
3
4 main(){
5     beta();
6 }

```

Fil 2:

```

1 extern int k;
2
3 alpha(){
4     int i = 4, l = 5;
5     i += k+1;
6 }

```

Fil 3:

```

1 extern int i, j;
2 extern alpha();
3
4 beta(){
5     int k = 6;
6     i = j+k;
7     alpha();
8 }

```

Da vet ikke kompilatoren hvilke adresser i C og D som er ledige.

Den kjenner heller ikke adressene til eksterne funksjoner og variable.

Relokerbar kode

Løsningen er å la hver rutine bruke adresser fra 0 samt legge ved *relokeringsinformasjon*. Så kan linkerens endre koden.

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 aksesserer som

D[0]+tillegg

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 }
```