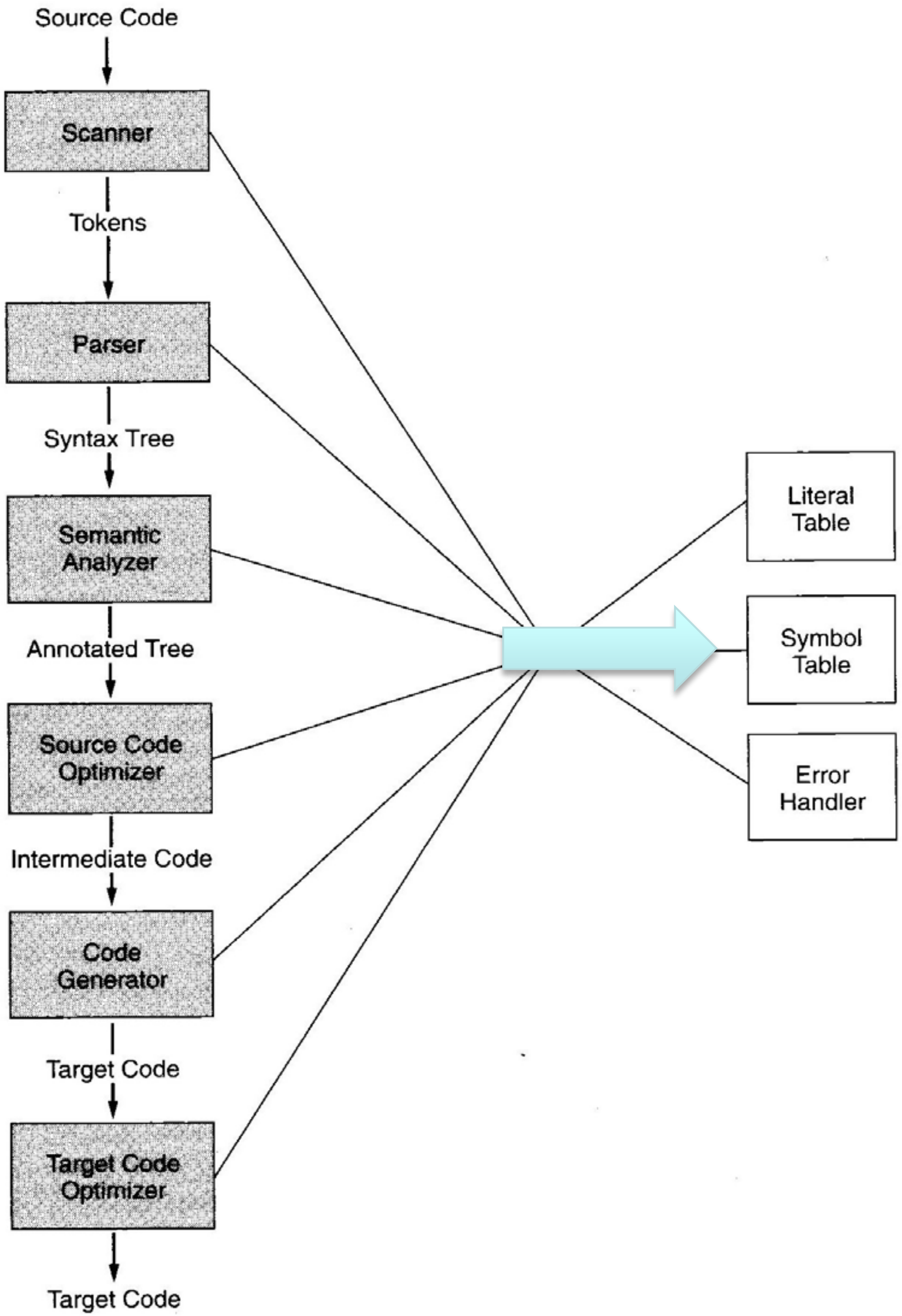
Semantisk Analyse del II

Symboltabellen

Kapittel 6.3



Deklarasjoner

- Konstant-deklarasjoner

  const int SIZE = 199; (C)

- Type-deklasjoner

  type Table = array [1..SIZE] og Entry; (Pascal)

- Variabel-deklarasjoner
- Prosedyre/funksjons-deklarasjoner
- Klasse-deklarasjoner
- I tillegg: Implisitte deklarasjoner («declaration by use»)

EN ELLER FLERE SYMBOLTABELLER?

Symboltabellen

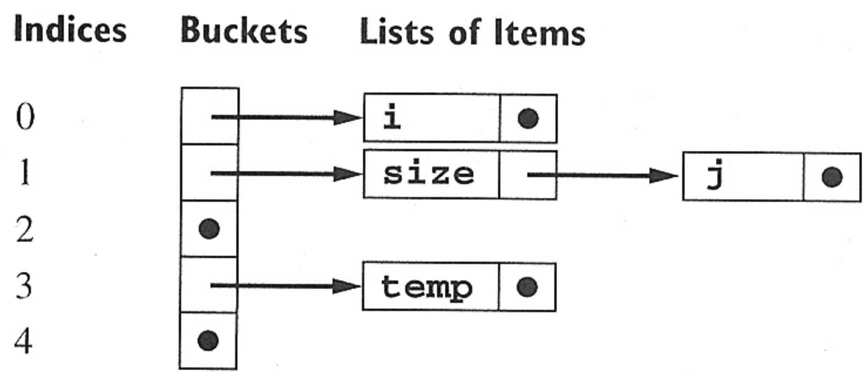
- Operasjoner:
  - lookup(id)
  - insert(id)
  - delete(id)

```
{ int i; ... double d;
  void p(...)
  { int i;
    ...
  }
  int j
}
```

- To hovedfilosofier for implemetasjon:
  - Selve syntakstreet
    - Insert/delete blir implisitte (etter hvordan man flytter seg i treet).
    - Lookup blir en lete-prosess (vanskelig å få effektiv?)
  - Tradisjonell søkestruktur
    - Lister
    - Søketrær
    - Hash-tabeller

Mye brukt: hashing

- Gir tilnærmet konstant tid for både oppslag, innsetting og sletting.




```
{
  int temp;
  int j;
  real i
  void size(...)
    { ...
    }
}
```

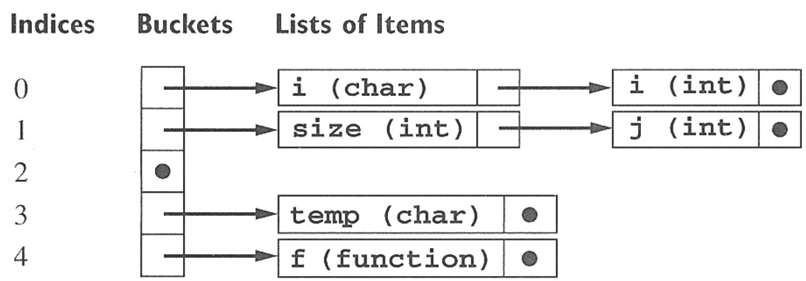
Blokkstruktur: eksempler

```
int i,j;

int f(int size)
{ char i, temp;
   ...
   { double j;
     ...
   }
   ...
   { char * j;
     ...
   }
}
```

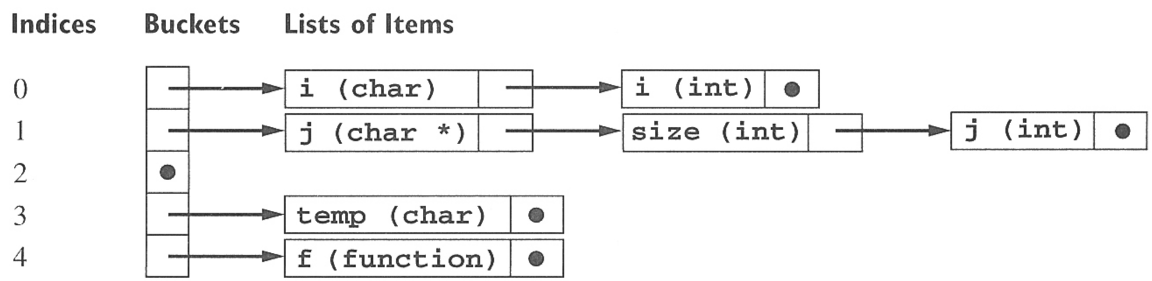
```
program Ex;
var i,j: integer;

function f(size: integer): integer;
var i,temp: char;

  procedure g;
  var j: real;
  begin
    ...
  end;

  procedure h;
  var j: ^char;
  begin
    ...
  end;

begin (* f *)
  ...
end;

begin (* main program *)
  ...
end.
```

Blokkstruktur: bruk av stakk

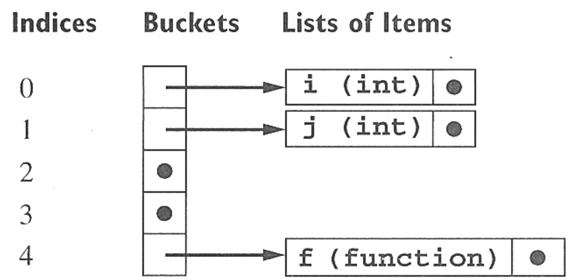
```
int i,j;

int f(int size)
{ char i, temp;
   ...
   { double j;
     ...
   }
   ...
   { char * j;
     ...
   }
}
```



(a) After processing the declarations of the body of f



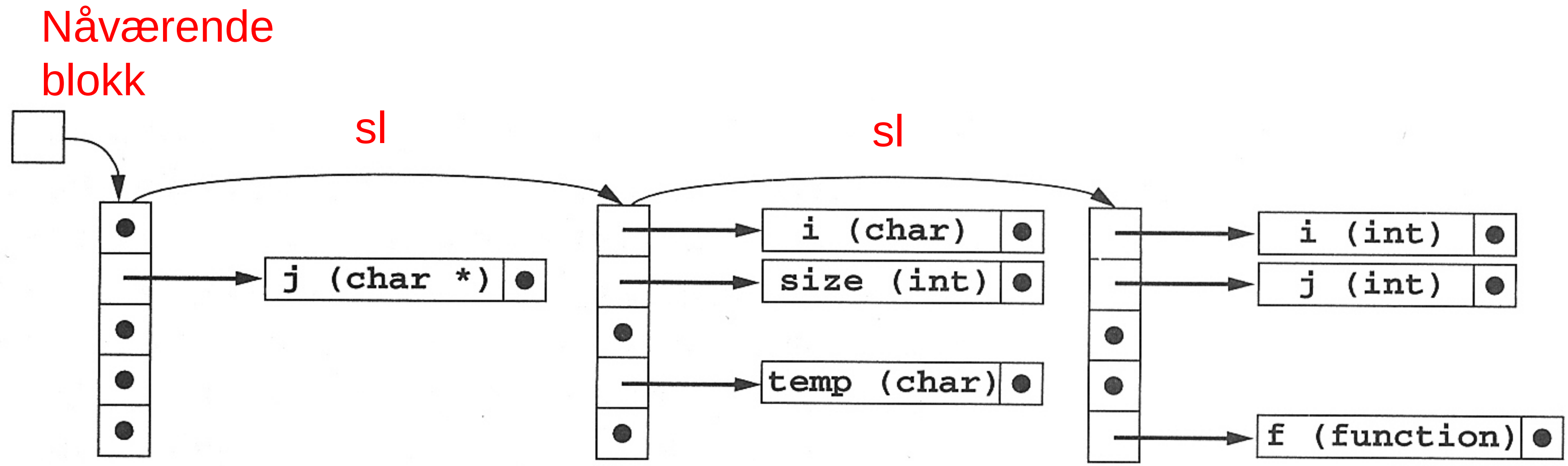
(b) After processing the declaration of the second nested compound statement within the body of f



(c) After exiting the body of f (and deleting its declarations)

Blokkstruktur: flere tabeller

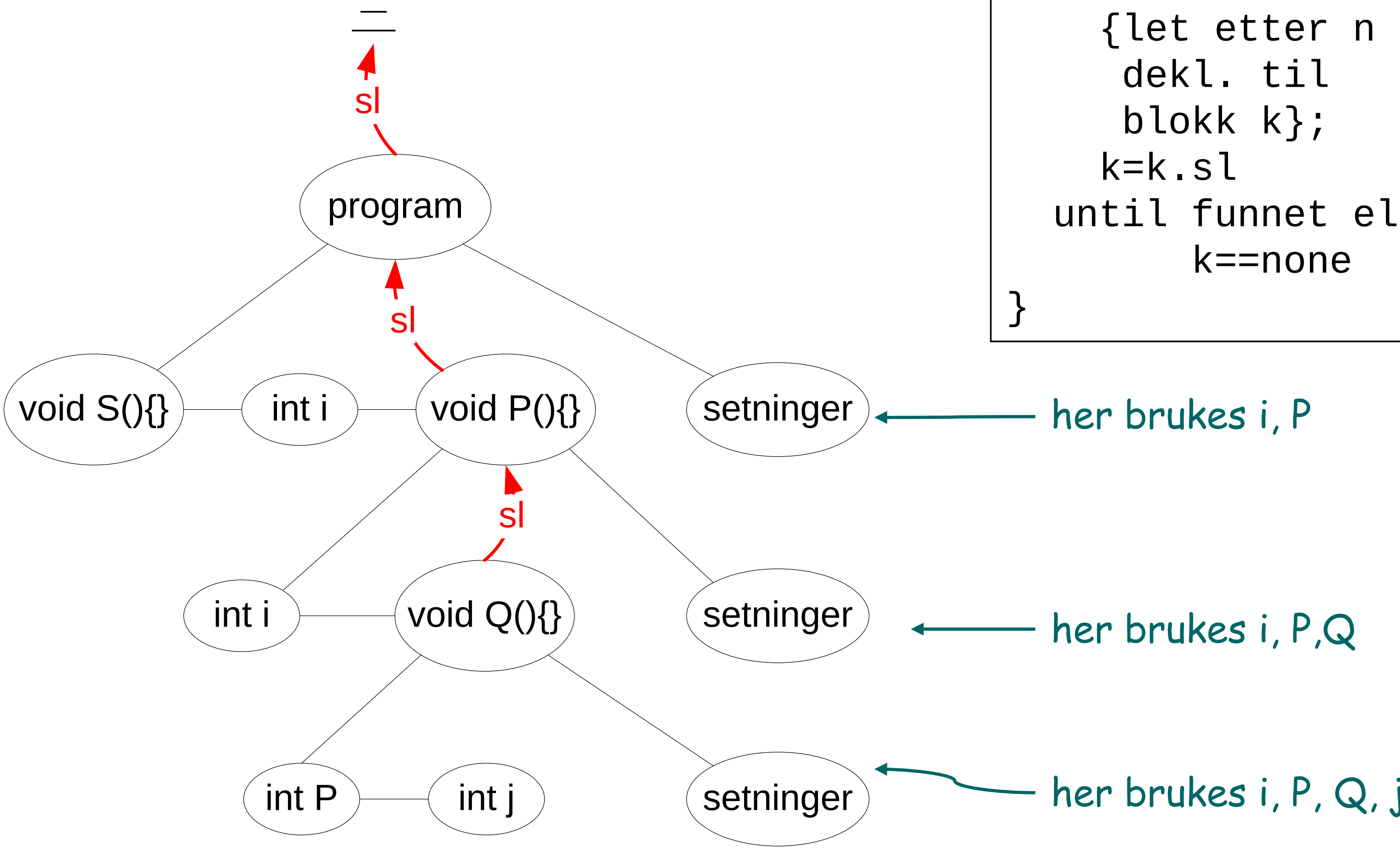
- Samler deklarasjonene i hver blokk i hver sin tabell.
- Bruker hashing innenfor hver tabell.
- Bruker statisk link pekere, og implementerer lookup ved leting.



INF 5110 - 2013

Bruk av syntakstre til lookup

```
lookup(n){
  k=nåværende blokk
  do
    {let etter n i
     dekl. til
     blokk k};
    k=k.sl
  until funnet eller
        k==none
}
```



Skop – problemstillinger

```
typedef int i;
int i;
```

```
int gcd(int n, int m)
{ if (m == 0) return n;
  else return gcd(m,n % m);
}
```

```
int i = 1;

void f(void)
{ int i = 2, j = i+1;
  ...
}
...
```

```
void f(void)
{... g() ...}

void g(void)
{... f() ...}
```

Sequential <> Collateral

Skop – problemstillinger forts

```
void g(void); /* function prototype
                           declaration */

void f(void)
{... g() ...}

void g(void)
{... f() ...}
```

Dynamisk skop: binde navn via dynamisk link

```
#include <stdio.h>

int i = 1;

void f(void)
{ printf("%d\n",i);}

void main(void)
{ int i = 2;
  f();
  return 0;
}
```

```
void Y(){
  int i;
  void p(){
    int i;
    ...
    Q();
    ...
  }
  void Q(){
    ...
    i = 5;
    ...
  }
  ...
  P();
  ...
}
```

hvilken 'i'
i
'i = 5'?

INF 5110 - 2013

Større eksempel – kapittel 6.3.5

- Attributtgrammatikk – definerer statisk semantikk, ikke implementasjon

$S \rightarrow exp$

$exp \rightarrow$ **(** exp **)** | exp **+** exp | ***id*** | ***num*** | **let** $dec\text{-}list$ **in** exp

$dec\text{-}list \rightarrow dec\text{-}list$ **,** $decl$ | $decl$

$decl \rightarrow$ ***id*** **=** exp

```
let x = 2, y = 3 in
  (let x = x+1, y=(let z=3 in x+y+z)
   in (x+y)
  )
```

exp:	symtab	arvet
	nestlevel	arvet
	err	syntetisert
declist:	intab	arvet
decl:	outtab	syntetisert
	nestlevel	arvet

insert(tab, name, l)	leverer ny tabell
isIn(tab, name)	ja/nei
lookup(tab, name)	gir nivået

Regler

1. Ikke samme navn på to i samme let-blokk

```
let x=2,x=3 in x+1
```

2. Navne må deklareres

```
let x=2 in x+y
```

3. 'Innermost' binding

```
let x=2 in (let x=3 in x)
```

4. 'sequential' deklarasjon

```
let x=2,y=x+1 in (let x=x+y,y=x+y in y)
```

Grammar Rule	Semantic Rules
$S \rightarrow exp$	$exp.symtab = emptytable$ $exp.nestlevel = 0$ $S.err = exp.err$
$exp_1 \rightarrow exp_2 + exp_3$	$exp_2.symtab = exp_1.symtab$ $exp_3.symtab = exp_1.symtab$ $exp_2.nestlevel = exp_1.nestlevel$ $exp_3.nestlevel = exp_1.nestlevel$ $exp_1.err = exp_2.err$ **or** $exp_3.err$
$exp_1 \rightarrow$ (exp_2)	$exp_2.symtab = exp_1.symtab$ $exp_2.nestlevel = exp_1.nestlevel$ $exp_1.err = exp_2.err$
$exp \rightarrow$ **id**	$exp.err =$ **not** $isin(exp.symtab,$**id**$.name)$ } 2
$exp \rightarrow$ **num**	$exp.err =$ **false**
$exp_1 \rightarrow$ **let** $dec\text{-}list$ **in** exp_2	$dec\text{-}list.intab = exp_1.symtab$ $dec\text{-}list.nestlevel = exp_1.nestlevel + 1$ $exp_2.symtab = dec\text{-}list.outtab$ } 3 $exp_2.nestlevel = dec\text{-}list.nestlevel$ $exp_1.err = (decl\text{-}list.outtab = errtab)$ **or** $exp_2.err$

INF 5110 - 2013

$dec\text{-}list_1 \rightarrow dec\text{-}list_2$ **,** $decl$	$dec\text{-}list_2.intab = dec\text{-}list_1.intab$ $dec\text{-}list_2.nestlevel = dec\text{-}list_1.nestlevel$ $decl.intab = dec\text{-}list_2.outtab$ $decl.nestlevel = dec\text{-}list_2.nestlevel$ $dec\text{-}list_1.outtab = decl.outtab$	4
$dec\text{-}list \rightarrow decl$	$decl.intab = dec\text{-}list.intab$ $decl.nestlevel = dec\text{-}list.nestlevel$ $dec\text{-}list.outtab = decl.outtab$	4
$decl \rightarrow$ **id** **=** exp	$exp.symtab = decl.intab$ $exp.nestlevel = decl.nestlevel$ $decl.outtab =$ **if** $(decl.intab = errtab)$ **or** $exp.err$ **then** $errtab$ **else if** $(lookup(decl.intab,$ **id** $.name) =$ $decl.nestlevel)$ **then** $errtab$ **else** $insert(decl.intab,$ **id** $.name, decl.nestlevel)$	1

Neste forelesning: Fredag 8. mars (seminarrom Sed)

OPPGAVER: 6.17 OG 6.18
START TYPESJEKKING