# SCOPING STATICO - DINAMICO

Definire brevemente scoping (statico e dinam.)

```
{ int a=0; int y=1;
  (*)
  void exec () { int x;
                 (**)
                 y = fun ();
                 a = a+1;
               }
  while (a<=1) { exec (); }
}
```

Completare il codice in modo tale che in caso di scoping statico le due esecuzioni di fun restituiscono lo stesso valore; in caso di scoping dinamico valori diversi

→ Dobbiamo definire fun();

→ Dobbiamo utilizzare una variabile non locale in fun() che abbia sia un ambiente nel blocco di definizione sia nel blocco di chiamata

Scegliamo di usare x → dobbiamo dichiararla e inizializzarla nel blocco di definizione

→ dobbiamo inizializzarla nel blocco di chiamata

(*) Blocco di def: `int x=2; int fun () { return x; }`

(**) Blocco di chiamata: x = 0

Scoping statico: fun() ha come ambiente di riferimento quello globale in cui x non viene modificata ⇒ restituisce sempre 2

Scoping dinamico: fun() ha come ambiente di riferimento quello dentro exec(), quindi x=0 ed essendo a una variabile globale vale 0 alla prima chiamata e 1 alla seconda.

---

## SCOPING + BINDING

↳ Def. brevemente scoping (statico / dinamico)

↳ Dire quando servono le regole di binding

Servono le regole di scoping quando una procedura ha un ambiente non locale.
Servono anche le regole di binding quando una procedura con ambiente non locale è passata come parametro ad un'altra procedura

↳ Definire le regole di binding (shallow / deep)
⇒ dire quali sono gli ambienti usati?

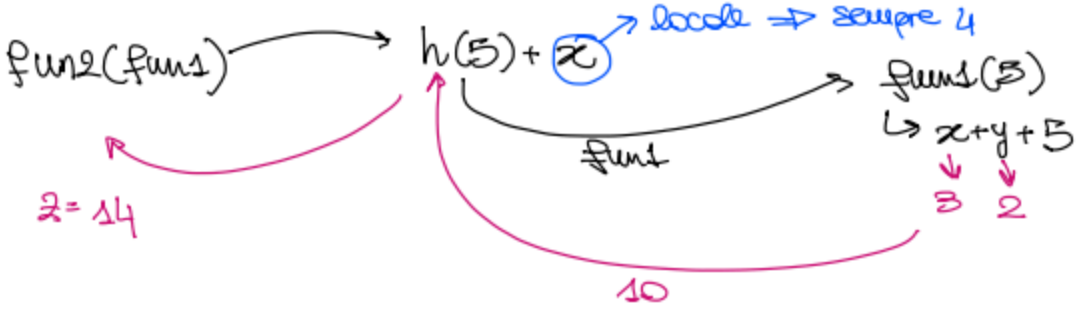
```
int x=3; int y=2;                          ① Ambiente del blocco di definizione
int fun1(int n){return x+y+n;}             (statico) → ambiente non locale → servono le regole di scoping

int fun2(int h(int b)){
    int x=4; int y=4;                      ③ Ambiente del blocco di chiamata effettiva mediante il parametro attuale
    return h(5)+x;                         (shallow / dinamico)
}
{int x=6; int y=3;                         ② Ambiente del blocco in cui viene creato il legame tra parametro attuale e formale
 int z=fun2(fun1);                         (deep / dinamico)
}
```

→ servono le regole di binding

fun2 non ha ambiente non locale → non utilizza

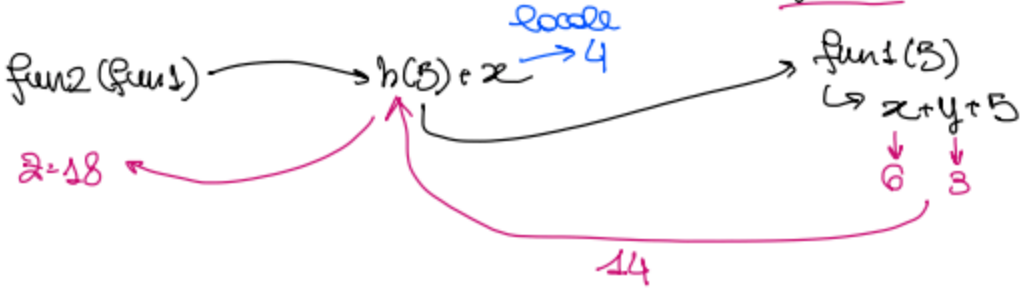
## SCOPING STATICO

fun1 → Ambiente di riferimento è ① → x=3, y=2 (supponiamo non ci siano modifiche)



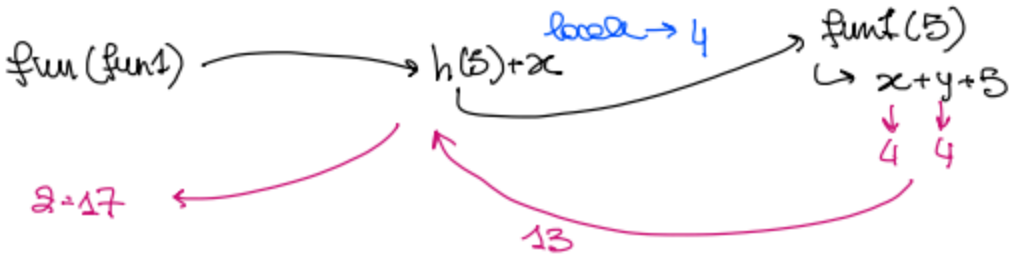

## SCOPING DINAMICO + DEEP BINDING

fun1 → Ambiente di riferimento è ② → x=6, y=3




## SCOPING DINAMICO + SHALLOW BINDING

fun1 → Ambiente di riferimento è ③ → x=4, y=4



# SCOPING : IMPLEMENTAZIONE

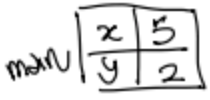
↳ Def brevemente scoping (statico/dinamico)
↳ Def link statico / catena statica
↳ Def CRT e come è approssimata

main:

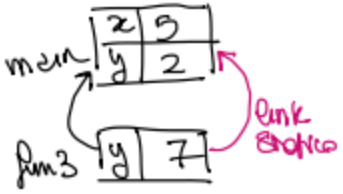
```
int x; int y;
void fun1() { int z = x+y; }
void fun2() { y = y+1 }
void fun3() { int y = x+2;
              fun2();
              fun1();
            }
x=5; y=2;
fun3();
}
```


Sd(main) = 0
Sd(fun1) = Sd(fun2) = Sd(fun3) = 1

## Scoping statico




chiamata main → fun3()




Per costruire il link statico dobbiamo calcolare (chiamante, chiamato)

$k_{fun3} = Sd(main) - Sd(fun3) + 1$
$= 0 - 1 + 1 = 0$

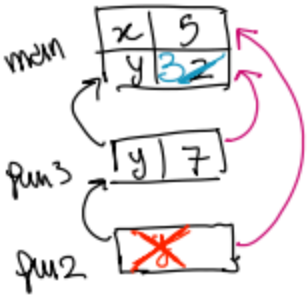
$CS(fun3) = main$

Esecuzione: $y = x + 2$ (locale)
↳ Per risolvere x dobbiamo calcolare $N_x$ (dobbiamo risalire di 1 la catena statica)

$N_x = Sd(fun3) - Sd(main) = 1$

Chiamata fun3 → fun2

procedura più vicina nella struttura di annidamento statica che def x




Costruiamo link statico:

$K_{fun2} = Sd(fun3) - Sd(fun2) + 1$

$= 1 - 1 + 1 = 1$

$CS(fun2) = CS(fun3) = main$

Esecuzione

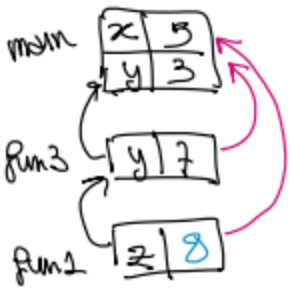
$y = y + 1$ → 3 y nel main

↳ 2

↳ $Ny = Sd(fun2) - Sd(main) = 1$

⇒ y è nel main = 2

Chiamata fun3 → fun1




Costruiamo link statico:

$K_{fun1} = Sd(fun3) - Sd(fun1) + 1$

$= 1$

$CS(fun1) = CS(fun3) = main$

Esecuzione di fun1 $z = x + y$ → 5, → 3 nel main

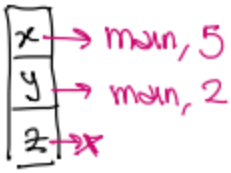
$N_x = Sd(fun1) - Sd(main) = 1$

$N_y = Sd(fun1) - Sd(main) = 1$

Termine tutto

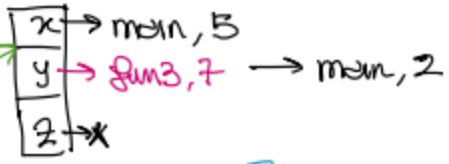
## Scoping dinamico

Evoluzione CRT



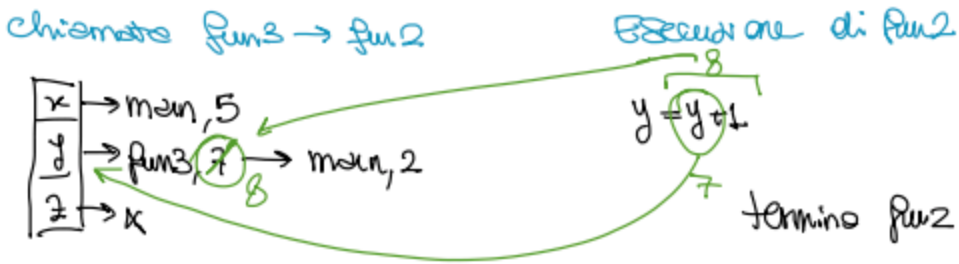

Esc. main

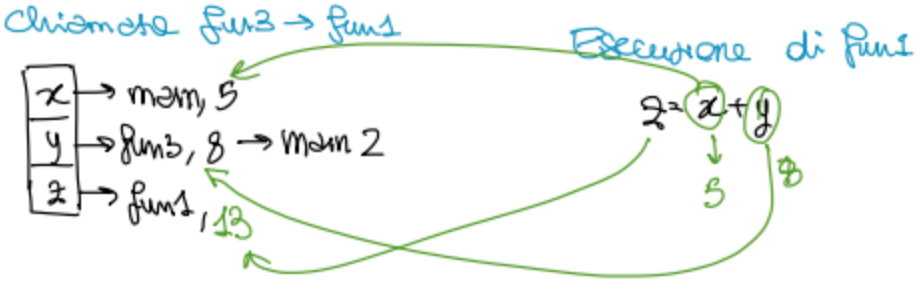
Chiamata main → fun3




Esecuzione fun3

$y = x + 2$



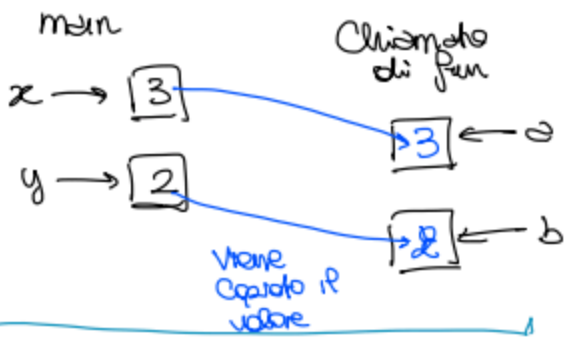

Termina tutto

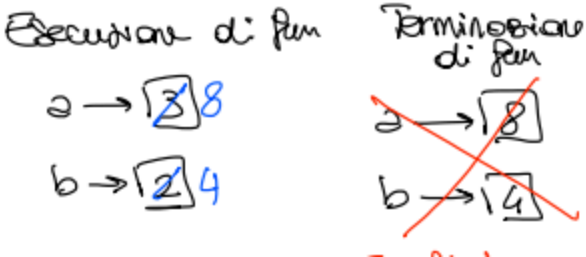
## PASSAGGIO DI PARAMETRI

↳ def brevemente le tipologie di passaggio di parametri citate nel testo

```
void fun(int a, int b){
    a = a+5;
    b = 2b;
}

void main (){
    int x=3; int y=2;
    fun(x,y);
}
```

Passaggio per valore

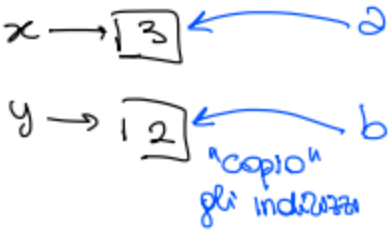


# Passaggio per riferimento

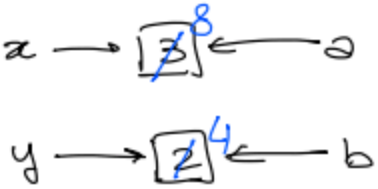
⇒ nel main i valori di x e y restano invariati

main

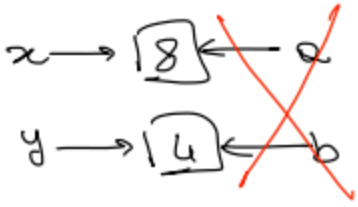



chiamata di fun

Esecuzione di fun




Terminazione di fun




Si eliminano a, b (indirizzi)

⇒ nel main i valori di x e y sono stati modificati

---

## RICORSIONE

↳ Def. brevemente ricorsione e ricorsione in coda

```
int function (lista list){
        if length(list)=0 then return 0;
        else return (car(list)+ function(cdr(list)));
}
```

non è in coda

function((3,6,9)) ← 18

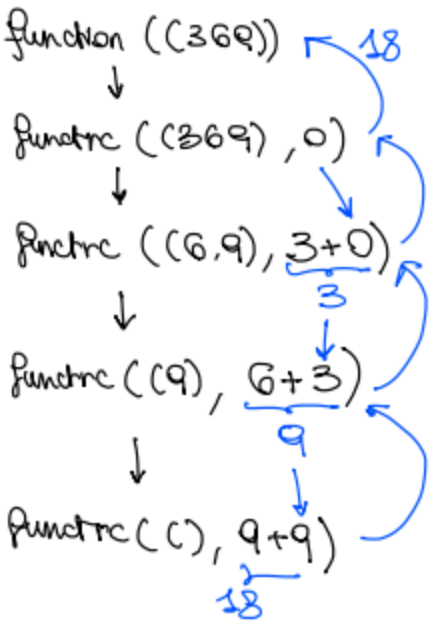
↳ 3 + function((6,9)) ← 15

↳ 6 + function((9)) ← 9

↳ 9 + function(()) ↺ 0

⇒ Il programma calcola ricorsivamente la somma

```
int function (lista list) { return functrc (list, 0); }

int functrc (lista list, int res) {
        if length (list) = 0 return res;
        else return functrc (cdr (list), car (list) + res));
        }
```




---

## SEMANTICA

$\rho = [x \mapsto 2, y \mapsto l_y]$ → cosa è un ambiente dinamico

$\sigma = [l_y \mapsto 4]$ → cosa è una memoria

$y := (x+4) * y$ descrivere le derivazioni della semantica dinamica

↳ due brevemente cosa è la sem. dinamica per assegnamento (comandi) e per espressioni

Dobbiamo eseguire il comando $y := (x+4) * y$

⇒ la regola chiedi di valutare l'espressione a destra

$$\frac{\rho \vdash \langle (x+4)*y, \sigma\rangle \rightarrow^* (k, \sigma)}{\rho \vdash \langle y := (x+4)*y, \sigma\rangle \rightarrow \langle y := k, \sigma\rangle}$$

⇒ Dobbiamo valutare l'espressione $(x+4) * y$

→ Valutazione di $(x+4) * y$

$$\frac{\rho \vdash \langle x+4, \sigma\rangle \rightarrow \langle k_1, \sigma\rangle}{\rho \vdash \langle (x+4)*y, \sigma\rangle \rightarrow \langle k_1 * y, \sigma\rangle}$$

⇒ Dobbiamo valutare $x+4$

→ Valutazione di $x+4$

$$\frac{\rho \vdash \langle x, \sigma\rangle \rightarrow \langle 2, \sigma\rangle \quad \rho(x) = 2}{\rho \vdash \langle x+4, \sigma\rangle \rightarrow \langle 2+4, \sigma\rangle}$$

$\rho \vdash \langle x+4, \sigma\rangle \rightarrow^* \langle 6, \sigma\rangle$

⇒ $k_1 = 6$

$$\rho \vdash \langle 2+4, \sigma\rangle \xrightarrow{k_1} \langle 6, \sigma\rangle$$

→ Valutazione di $6 * y$

$$\frac{\rho \vdash \langle y, \sigma\rangle \rightarrow \langle 4, \sigma\rangle \quad \rho(y) = l_y \wedge \sigma(l_y) = 4}{\rho \vdash \langle 6*y, \sigma\rangle \rightarrow \langle 6*4, \sigma\rangle}$$

$$\rho \vdash \langle 6*4, \sigma\rangle \xrightarrow{k} \langle 24, \sigma\rangle$$

$$\frac{\rho \vdash \langle (x+4)*y, \sigma\rangle \to^{*} \langle 24, \sigma\rangle}{\rho \vdash \langle y := (x+4)*y, \sigma\rangle \to \langle y := 24, \sigma\rangle}$$

Dobbiamo eseguire $y := 24$

$$\rho \vdash \langle y := 24, \sigma\rangle \to \sigma[\ell_y \mapsto 24], \quad \rho(y) = \ell_y$$

---

Derivazione semantica statica (supponiamo che venisse chiesto la sem. statica)

$\Delta = [y \mapsto \text{int loc}, x \mapsto \text{int}]$ → cosa è ambiente statico

Dobbiamo verificare se $y := (x+4)*y$ è ben formato

$$\frac{\Delta \vdash (x+4)*y : e}{\Delta \vdash y := (x+4)*y}, \quad \Delta(y) = e\ \text{loc}$$

int

Sappiamo da $\Delta$ che $\Delta(y) = \text{int loc}$ dobbiamo verificare che l'espressione sia int

Dobbiamo verificare $\Delta \vdash (x+4)*y : \text{int}$

$$\frac{\Delta \vdash x+4 : \text{int} \qquad \Delta \vdash y : \text{int}}{\Delta \vdash (x+4)*y : \text{int}}$$

vero perché $\Delta(y) = \text{int loc}$

è int se

Dobbiamo verificare $\Delta \vdash x+4 : \text{int}$

vero $\Delta(x) = \text{int}$

$$\frac{\Delta \vdash x : \text{int} \qquad \Delta \vdash 4 : \text{int}}{\Delta \vdash x+4 : \text{int}}$$

vero perché assumo

Verificato

Fare Semantica: $\rho \vdash y := 3x+y$ sem. dinamica $\rho = [x \mapsto 3, y \mapsto l_y]$ $\sigma = [l_y \mapsto 2]$

$\rho \vdash$ if $(x>0)$ then $x := x+1$ else $x := x-1$

$\rho = [x \mapsto l_x]$ $\sigma = [l_x \mapsto 2]$

Fare ricorsivi:

```
liste fun(liste list){
    if length(list)=0 then return();
    else return concat(2*car(list), fun(cdr(list)));
}

su (369)
```

Fare scoping+binding:

```
{int a=4; int b=3;                                   (1)
 int pippo(int n){ return a+b+n;}
 int pluto(int f(int z)){
     (3) int a=6;
         return f(b)+a;
 }
 int exec(){
     (2) int b=2;
         return pluto(pippo);
 }
 int z = exec();
}
```

(1) a=4 → pippo
b=3 → pluto

(2) pluto → b=2
→ a=4 locale

pippo → b=2
→ a=4

(3) pluto (è re suo)
pippo → a=6
→ b=2

passato come parametro con ambiente non locale ⇓ regola di binding

ha ambiente non locale ↓ regola di scoping