Titolo nota 19/10/2015

ASSEMBLEA di ATENEO SOSPENSIONE DIDATTICA dalle 10:30 - 12:30

DOMANI MARTEDÌ 20/10/2015

GIOVEDÌ alle 15:00 RICEVIMENTO UFFICIO DIPARTIMENTO

o ——— o

Titolo nota 19/10/2015

MODELLO DI STATO

- Due pile: PILA AMBIENTE

contiene associazioni tra identificatori (nomi di variabili) e INDIRIZZI DI MEMORIA (Locazioni)

PILA MEMORIA

contiene associazioni tra locazioni e valori (vedremo che anche le locazioni stesse sono "valori" associabili in memoria)

```
int x = 10;
```

AMBIENTE

MEMORIA

Titolo nota 19/10/2015

FRAME AMBIENTE

$\varphi : Ide \to Loc_\perp$

"f"

FRAME MEMORIA

$\nu : Loc \to (\mathbb{N} \cup B \cup Loc)_\perp$

$Val_\perp$

"μ"

P: PILA AMBIENTE

$P = \{\Omega\} \cup \{\varphi . \rho \mid \varphi : Ide \to Loc_\perp , \rho \in P\}$

PILE AMBIENTE

M: PILA MEMORIA

$M = \{\Omega\} \cup \{\nu . \mu \mid \nu : Loc \to Val_\perp , \mu \in M\}$

PILE MEMORIA

$\rho \in P$

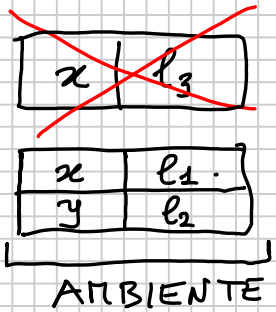
$\varphi : Ide \to Loc_\perp$

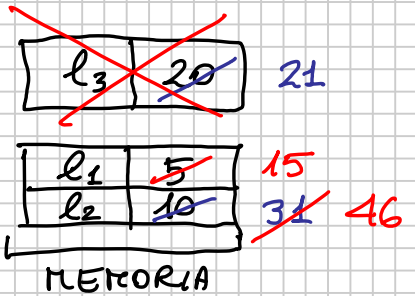
$\mu \in M$

$\nu : Loc \to Val_\perp$

Esempio : gestione di blocchi annidati rispetto al nuovo stato

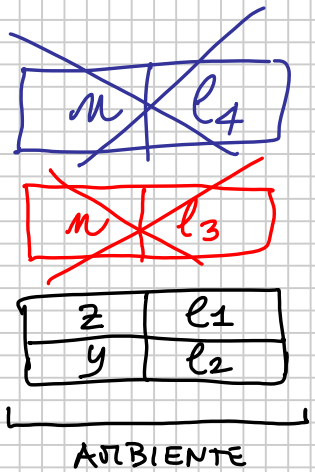
```
{ int x = 5;
  int y = 10;
  x = x + y;
  { int x = 20;
    x = x + 1;
    y = x + y;
  }
  y = x + y;
}
```

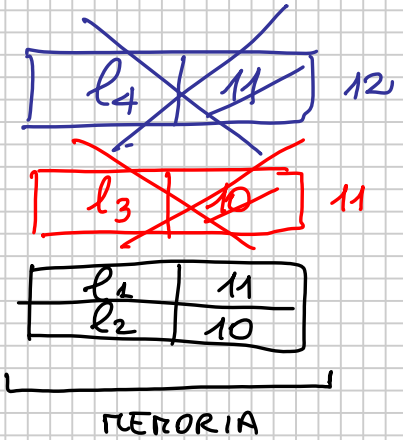




```
void incr (int n)
  { n = n+1; }

main ()
  { int y = 10;
    int z = 11;
①   incr (y);
②   incr (z);
  }
```





Il passaggio è PER VALORE!
Come possiamo SIMULARE il passaggio per VARIABILE? Usiamo i PUNTATORI

Titolo nota 19/10/2015

PUNTATORI in C

L'introduzione di ambiente e memoria ci ha consentito di esplicitare gli indirizzi (locazioni) associati alle variabili. In C gli indirizzi sono VALORI come tutti gli altri (possono essere associati a variabili di tipo PUNTATORE).

(1) DICHIARAZIONI di PUNTATORI

```
int *p;
```

i valori che p potrà assumere durante l'esecuzione sono INDIRIZZI di variabili di tipo int

```
int x;
```

i valori che x potrà assumere durante l'esecuzione sono INTERI

```
{ int x = 10;
  int *p;
  p = &x;
  x = x+1;
```

→ denota l'indirizzo della variabile x

x è una variabile di tipo int, &x è l'indirizzo di una variabile di tipo intero e come tale è un valore che può essere attribuito ad una variabile di tipo int * come p.

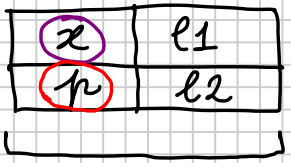
~~p = x+1;~~

NO! perché l'espressione x+1 ha un valore di tipo int, non int * !!

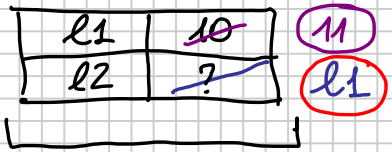
&x "sta per" la locazione associata ad x nella pila ambiente

~~x = &p;~~

NO! perché &p ha come valore un indirizzo, non int



AMBIENTE



MEMORIA

Titolo nota 19/10/2015

Come facciamo ad "usare" variabili di tipo puntatore?

Se p è una variabile di tipo **int ***

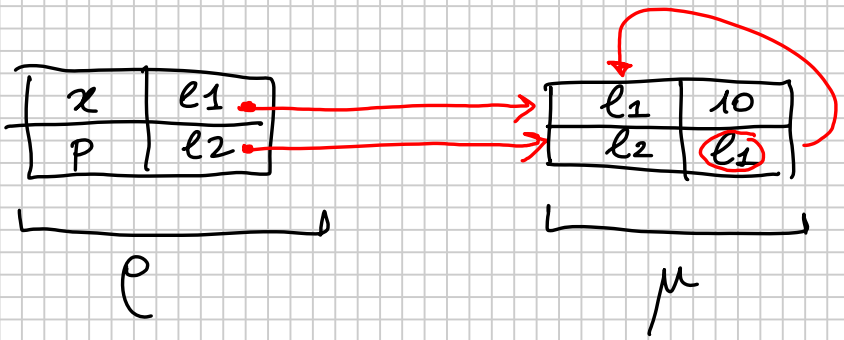
$*p$ "sta per la variabile (di tipo int) il cui indirizzo è il valore di p"

dereferencing (dereferenziazione)

TERMINOLOGIA: se p è una variabile di tipo puntatore e il valore di p è l'indirizzo di x, diciamo che

p PUNTA a x

x È PUNTATA da p




```
p = &x;
```

```
{ int x = 10;
  int y = 20;
  int *p = &x;
  int *q = &y;
```

nello stato risultante da queste dichiarazioni
x e *p sono "sinonimi"
y e *q " "

Sono equivalenti in questo stato

```
*p = *q + 1;
(x = y + 1;)
```

Sono equivalenti nel nuovo stato!

```
p = &y;
*p = *q + 1;
(y = y + 1;)
```

x	l_1
p	l_2 •
q	l_3 •
y	l_4 –

ρ

l_2	~~l_1~~ •	l_4
l_3 •	l_4 •	
l_4	~~20~~ •	21
l_1	~~10~~	21

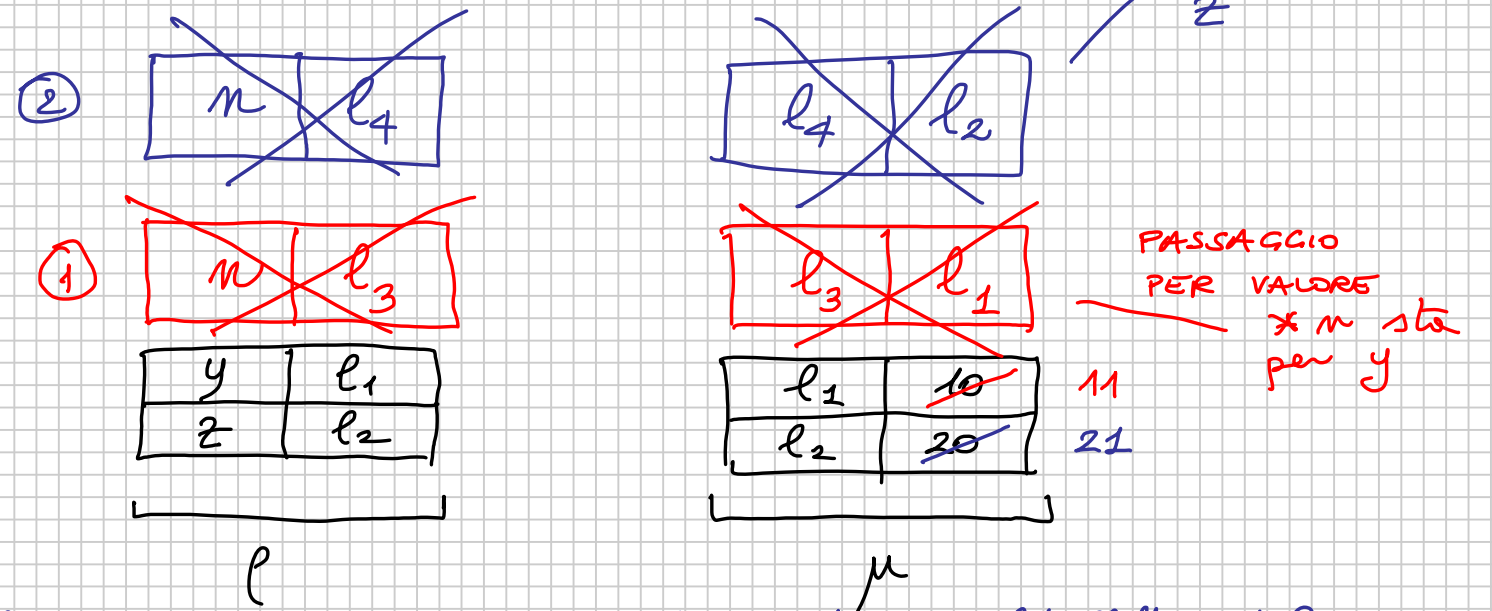
μ

Titolo nota 19/10/2015

PASSAGGIO per VARIABILE : SIMULAZIONE utilizzando i PUNTATORI

```
void incr (int *n)
  { *n = *n + 1; }

main ()
  { int y = 10;
    int z = 20;
①   incr(&y);
②   incr(&z);
  }
```



L'uso di parametri formali di tipo PUNTATORE consente di ottenere l'effetto del PASSAGGIO per VARIABILE anche in C (dove il PASSAGGIO dei PARAMETRI è SOLO PER VALORE)

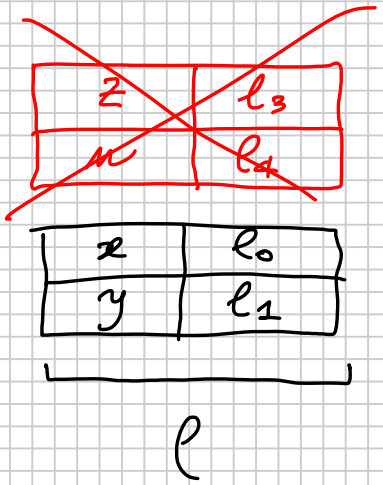
Titolo nota 19/10/2015

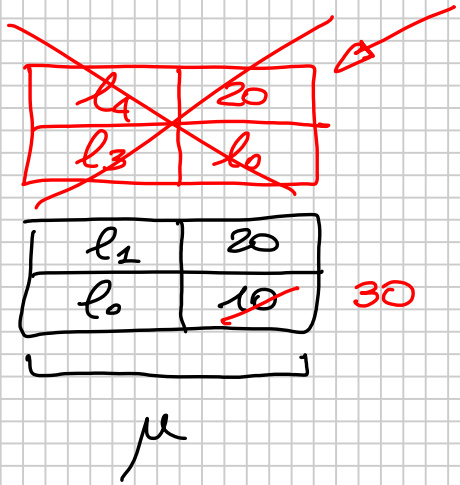
PROCEDURA con 2 PARAMETRI : z variabile da modificare, n valore con cui incrementare il valore di z

```
void incr (int *z , int n)
  {*z = *z + n;}

main( )
  { int x = 10 ; int y = 20;
    incr(&x, y);
```

↳ vorrei "aumentare il valore di x del valore di y"





REGOLE di SCOPING

```
void mcd (int x, int y)
  { ⋮ }

main ()
  { int z; int w;
    ⋮
    mcd(z,w);
    ⋮
  }
```

è come se la chiamata fosse "rimpiazzata" da

```
{ int x = z;
  int y = w;
  <corpo>
}
```

NON È ESATTAMENTE COSÌ!

```
int n = 20;
void incr (int *z)
  { *z = *z + n; }

main ()
  { int x = 30;
    int n = 150;
    incr(&x);
```

SCOPING STATICO: n è sempre questo!!

SCOPING DINAMICO

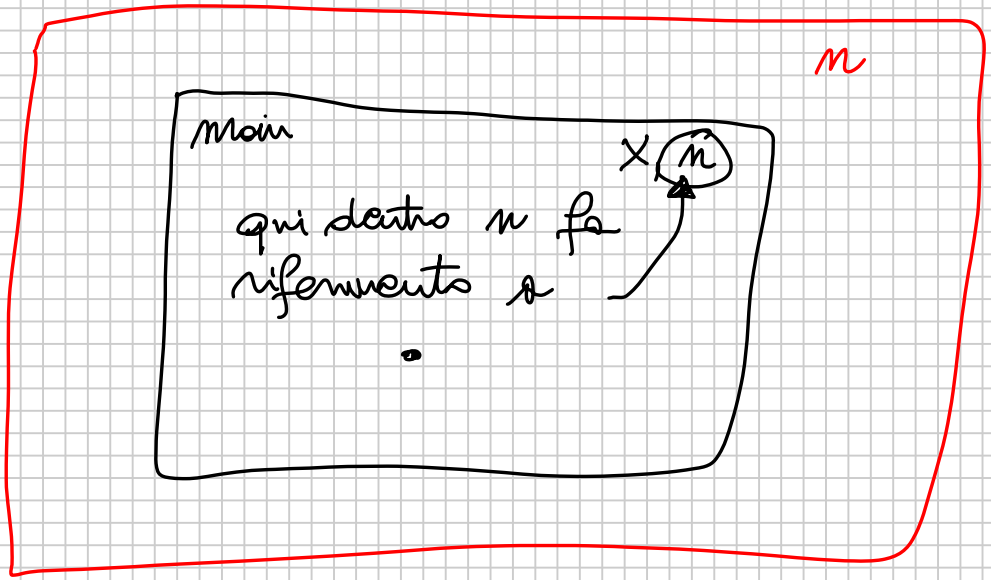
```
    { int *z = &x;
      *z = *z + n;
    }
```

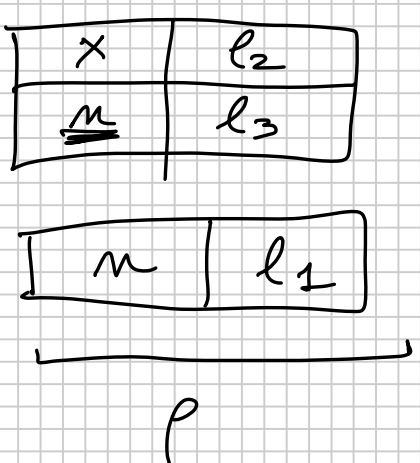
NO!!

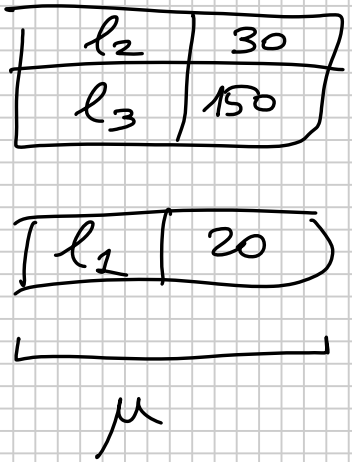
quale è il valore di x dopo questa chiamata?

Le eventuali variabili GLOBALI usate nella procedura sono SEMPRE quelle "vive" al momento della dichiarazione della procedura

Titolo nota 19/10/2015







Titolo nota 19/10/2015

SCOPING STATICO : i nomi globali (non locali, diversi dai parametri formali e da nomi dichiarati nel corpo della procedura/funzione) si riferiscono sempre (in ogni chiamata) a quelli presenti nell'AMBIENTE al momento della dichiarazione della procedura/funzione

SCOPING DINAMICO : i nomi globali si riferiscono a quelli presenti nell'ambiente al momento della CHIAMATA di procedura/funzione

```
int n = 20;
void incr (int *z)
  { *z = *z + n; }

{ int x = 10; int y = 20;
    { int n = 150;
      incr (&x);
    }
      :
    { int n = 3000;
      incr (&y);
    }
}
```

SCOPING STATICO
valore di x è 30

SCOPING DINAMICO
valore di x è 160

valore di y è 40

il valore di y è 3020