Titolo nota 24/09/2015

$Sem_e : Exp \times P \times M \to Val_\perp$

- $Sem_e(x, \rho, \mu) = \mu(\rho(x))$ valore che sta in memoria alla locazione di x

---

$Sem_d : Dec \times P \times M \to P \times M$

$Sem_d(T\ x = e;, \rho, \mu) = (\rho[^l/_x]^{add}, \mu[^v/_l]^{add})$

dove

$v = Sem_e(e, \rho, \mu)$

$l = succloc(\mu)$

$succloc : M \to Loc$

$$Sem_c : Com \times P \times M \to M$$

Titolo nota 24/09/2015

$$Sem_c(x = e_i, \rho, \mu) = \mu\left[{}^{v}/_{\rho(x)}\right]^{mod}$$

dove

$$v = Sem_e(e, \rho, \mu)$$

⋮

$$Sem_c(\{\underline{\underline{dl}} \quad cl\}, \rho, \mu) = \mu'$$

dove

$$(\varphi.\rho, \nu.\mu) = Sem_{dl}(dl, \omega.\rho, \omega.\mu)$$

$$\nu'.\mu' = Sem_{cl}(cl, \varphi.\rho, \nu.\mu)$$

$$\omega(m) = \perp$$

Titolo nota 24/09/2015

$$Sem_{cl}(c, \rho, \mu) = Sem_c(c, \rho, \mu)$$

$$Sem_{cl}(c\ cl, \rho, \mu) = \mu''$$

dove

$$\mu' = Sem_c(c, \rho, \mu)$$

$$\mu'' = Sem_{cl}(cl, \rho, \mu')$$

$$Sem_{cl}(c\ cl, \rho, \mu) = Sem_{cl}(cl, \rho, \mu')$$

dove

$$\mu' = Sem_c(c, \rho, \mu)$$

Com_list → Com | Com Com_list

Titolo nota 24/09/2015

$Sem_{dl} : Dec_list \times P \times M \to P \times M$

$Sem_{dl}(d, \rho, \mu) = Sem_d(d, \rho, \mu)$

$Sem_{dl}(d\ dl, \rho, \mu) = \sout{Sem_{dl}(dl, \rho', \mu')} \to (\rho'', \mu'')$

dove

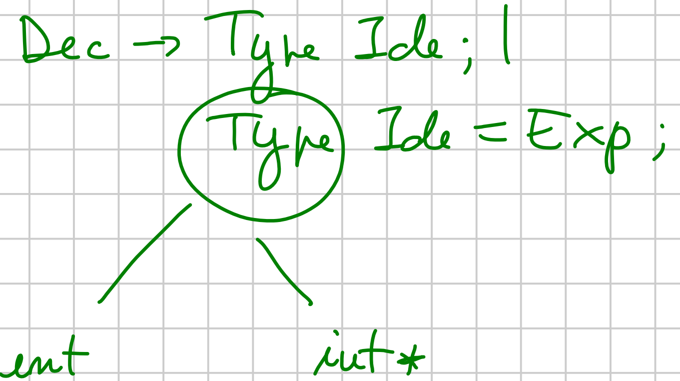
$(\rho', \mu') = Sem_d(d, \rho, \mu)$

$(\rho'', \mu'') = Sem_{dl}(dl, \rho', \mu')$

$Dec_list \to Dec \mid Dec\ Dec_list$

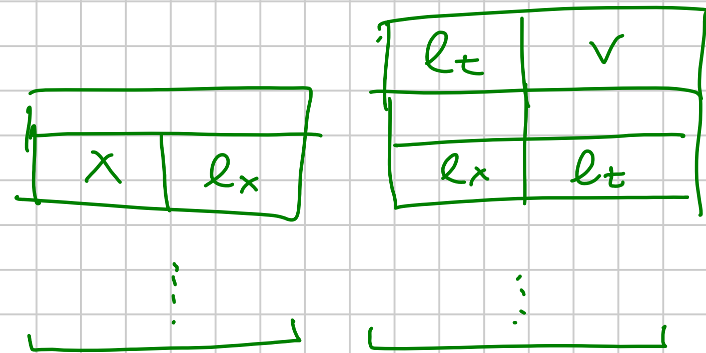
Exp → .... | *Ide | &Ide

Com → .... | *Ide = Exp;




$$Sem_e(*x, \rho, \mu) = \mu(\mu(\rho(x)))$$

$\rho(x) = l_x$, $\mu(l_x) = l_t$, $\mu(l_t) = v$



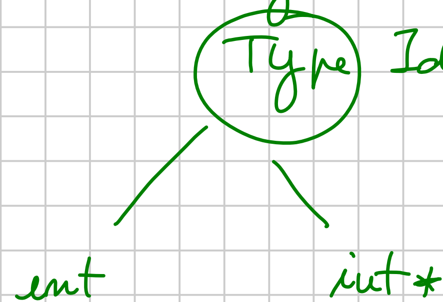
Titolo nota 24/09/2015

Exp → ..... | *Ide | &Ide

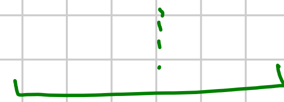
Com → ..... | *Ide = Exp;

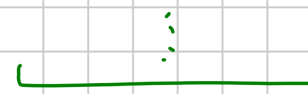
Dec → Type Ide; |

Type Ide = Exp;




$Sem_e(\&x, \rho, \mu) = \underbrace{\rho(x)}_{l_x}$



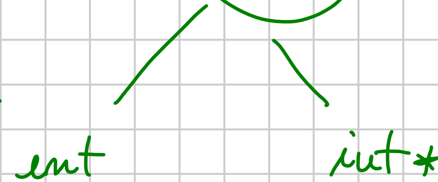



$Exp \to \ldots \mid *Ide \mid \& Ide$

$Com \to \ldots \mid *Ide = Exp;$

Dec → Type Ide; |

Type Ide = Exp;




$Sem_c(*x = e;, \rho, \mu) = \mu[^{m}/_{\mu(\rho(x))}]$ mod

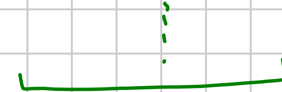
$\rho(x)$: $l_x$; $\mu(\rho(x))$: $l_t$

dove

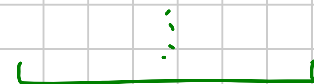
$m = Sem_e(e, \rho, \mu)$

$\mu(\mu(\rho(x))) = v$

| X | $l_x$ |
|---|---|



| $l_t$ | v |
|---|---|
| $l_x$ | $l_t$ |



# MEMORIA DINAMICA (HEAP)

La pile di memoria è "statica" in quanto si può capire quanta memoria verrà utilizzata per l'esecuzione di un blocco

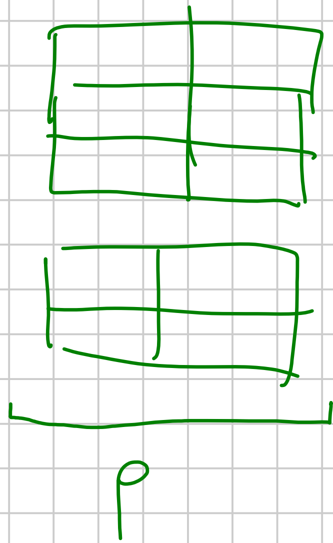
```
{
    int x = 5;
    int y = 10;
    x := x+1;
    :
}
```

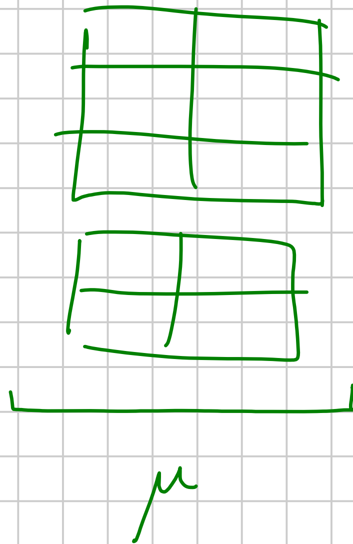
⟹ Verrà creato, per l'esecuzione del blocco un "frame" di memoria per memorizzare "due interi".

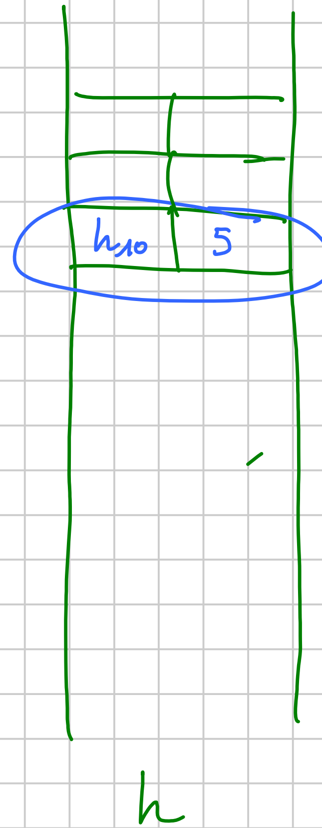
Titolo nota 24/09/2015

Esiste una espressione particolare per la creazione della memoria dinamica

→ creazione di una nuova associazione (locazione, valore)



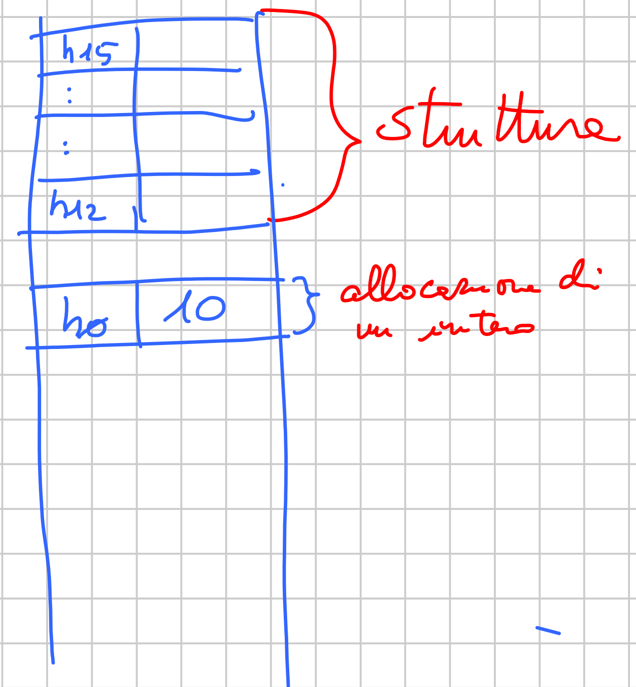




La memoria dinamica **non è** gestita come una pila.
La sua allocazione è indipendente dalla esecuzione dei blocchi ed è allocata esplicitamente dal programma

heap

espressione per allocare la memoria dinamica:

vogli allocare nella memoria dinamica lo spazio necessario per contenere una struttura (lo spazio è diverso a seconda della struttura da voleto allocare)

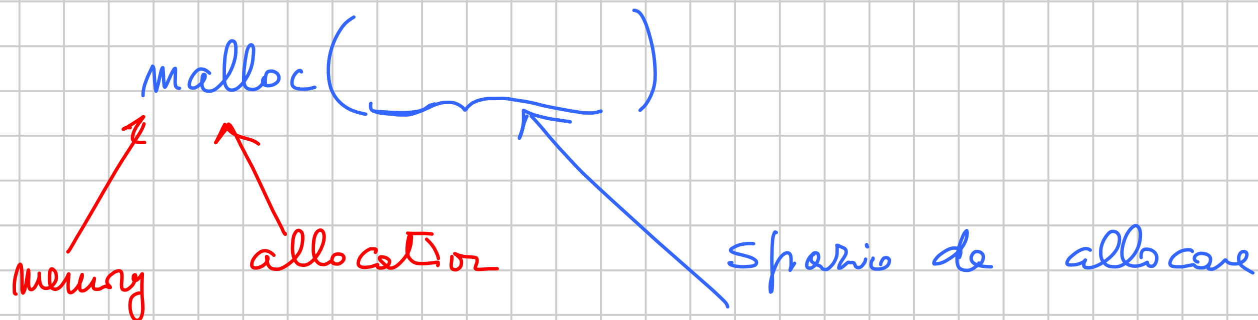



spazio da allocare:

int una parola di memoria

array lo spazio dipende dalla dimensione dell'array.

in C l'espressione particolare si chiama




qual'è la semantica di malloc( )

- crea in memoria heap lo spazio per memorizzare la struttura
- restituisce l'indirizzo, in memoria heap, del primo elemento dello spazio allocato

Titolo nota 24/09/2015

# Memoria dinamica

malloc ( )

spazio da allocare

se avessimo da allocare solo interi questo argomento non servirebbe.

esattamente quello che facciamo nel nostro "sottoinsieme didattico"

new :

- crea una associazione in memoria heap
- restituisce l'indirizzo della nuova associazione

Titolo nota — 24/09/2015

$\ell \in Loc^{\mu}$

$h \in Loc^{\zeta}$

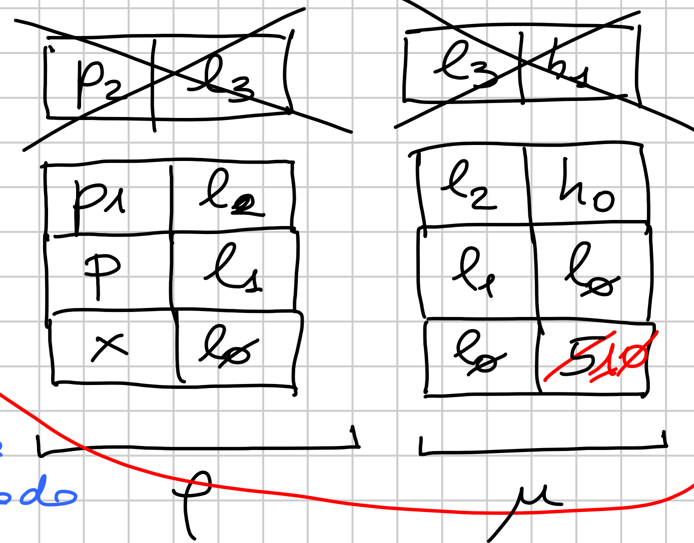
La memoria heap é indipendente dell'esecuzione dei blocchi

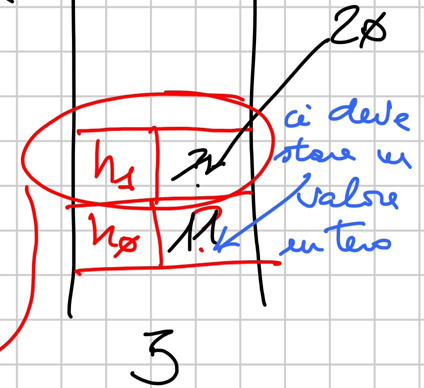
```
{ int x = 5;
  int* p = &x;
  int* p1 = new;   <- h0
  *p = 10;
  *p1 = 11;
  { int* p2 = new;
    *p2 = 20;
  }
```

• crea una nuova associazione in $\zeta$
• restituisca la nuova locazione

← la memoria alla locazione $h_1$ é occupata ma non c'é nessun modo per accedervi

garbage



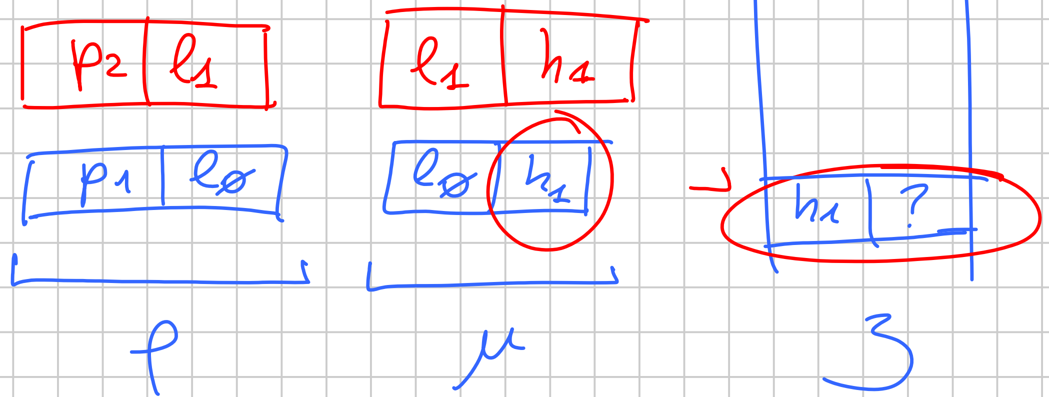



Titolo nota: Comando free — 24/09/2015

```
{ int* p1 = new;
free (p1);
{ int* p2 = new;
```

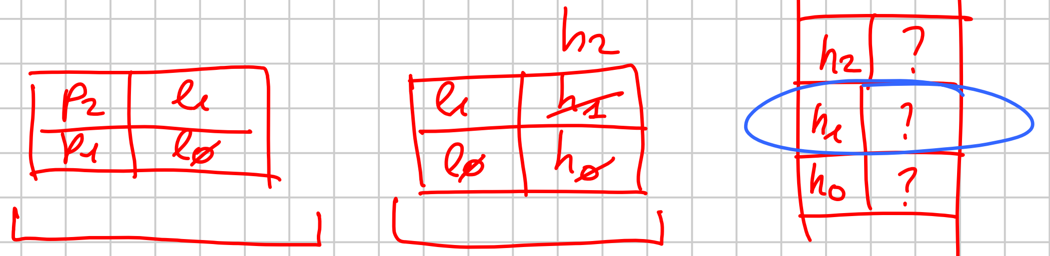
Valore che punta a una parola di memoria "cancellata"

"dangling reference"



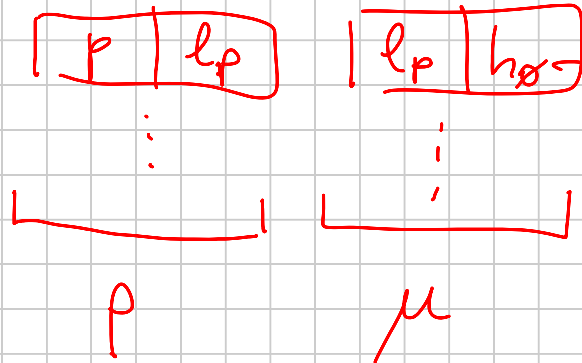
Se posso accedere alla memoria heap solo attraverso variabili puntatore, posso accedere a tante parole di memoria heap quante sono le variabili puntatore:

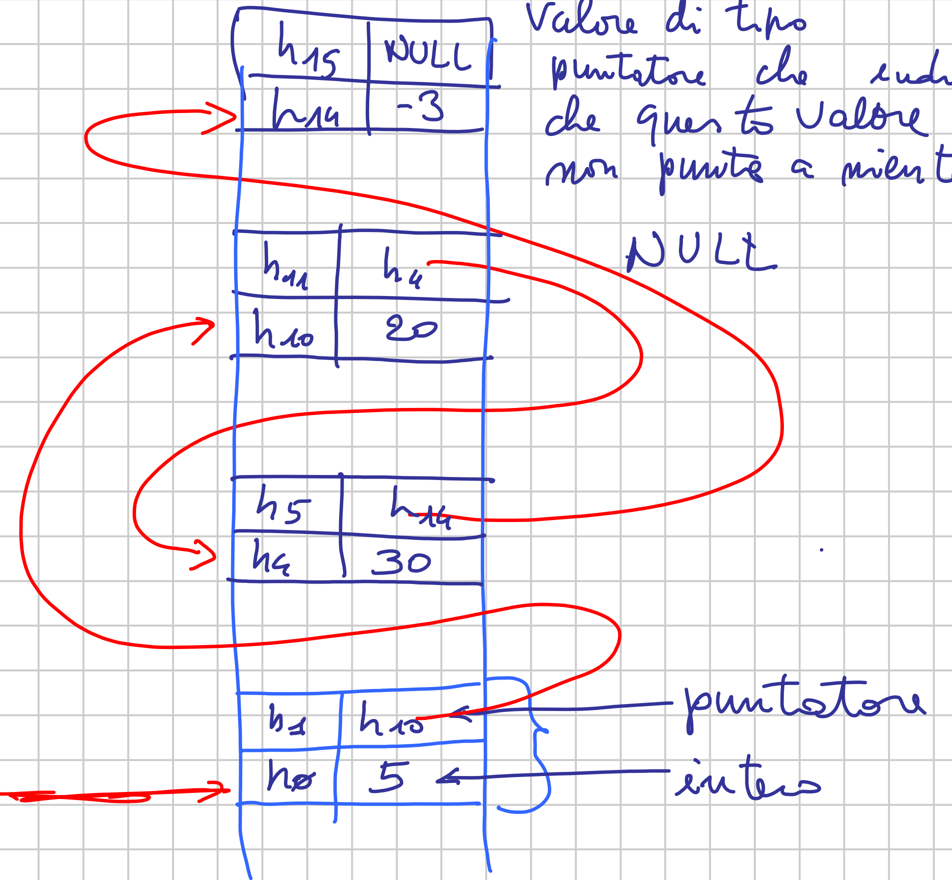
```
{
    int* p1 = new;
    int *p2 = new;
    p2 = new;
```



# STRUTTURA DINAMICA

## LISTA



Titolo nota 24/09/2015

Scrivere una funzione C con intestazione

int check (int a[], int dim)

che controlla che nell'array a ci sia almeno un valore ripetuto più di una volta. Restituisce 1 se tale valore esiste, ∅ altrimenti

| a | 1 | 2 | 5 | 5 | 1 | 5 | → 1 |
|---|---|---|---|---|---|---|---|
| a | -3 | 7 | 2 | 5 | 21 | 10 | → ∅ |

Titolo nota 24/09/2015

```
int check (int a [], int dim)
{ int i = 0;
  int trovato = 0;
  while (i < dim-1 && !trovato)
  { int j = i+1;
    int doppio = 0;
    while (j < dim && !doppio)
        if (a[j] == a[i]) doppio = 1;
          else j = j+1;
    if (doppio) trovato = 1;
       else i = i+1;
  }
return trovato;
}
```