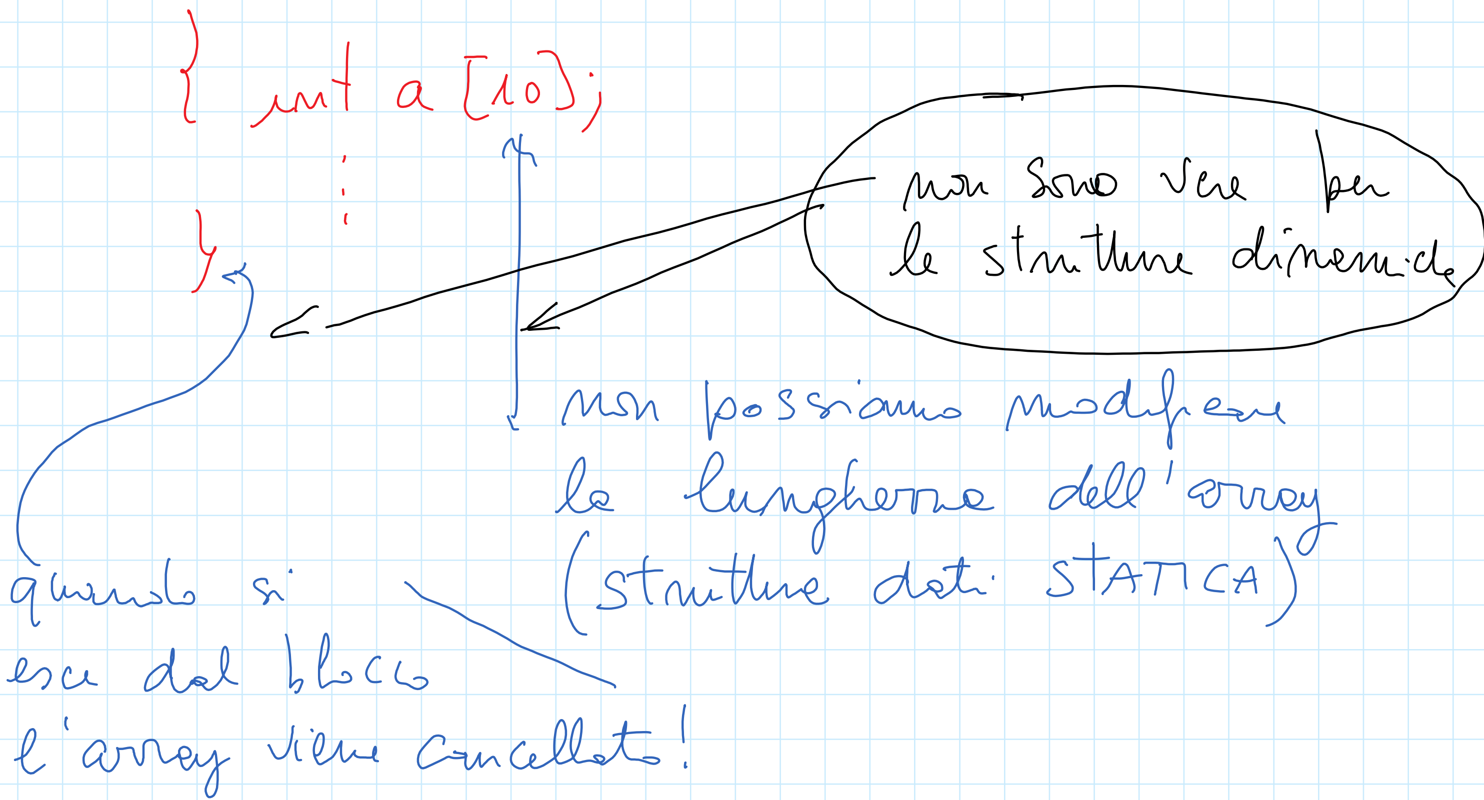
lunedì 21 ottobre 2019 09:15

Strutture dinamiche

Sono strutture dati che possono modificare le loro dimensione (aggiungendo o togliendo elementi) durante l'esecuzione dei programmi (indipendentemente dalla struttura a blocchi dei programmi stessi)



Come vengono gestite le strutture dinamiche?
Supponiamo di volerle gestire nella memoria a pila

```
{ int x = 5;
  dinamica a [10];
  int y = 10;
  aggiungo un elemento ad a;
```

y	l_{12}
a	l_1
x	l_0

cambiare l'ind.

spostare di una posizione

l_{12}	10
l_{11}	
l_{10}	
⋮	
l_2	
l_1	l_2
l_0	5

← nuovo elemento

non è gestibile una struttura dinamica sulla memoria a pila!!

lunedì 21 ottobre 2019 09:25

Soluzione: si dedica alle strutture dinamiche una nuova memoria (MEMORIA HEAP)

Si aggiunge quindi allo stato astratto (oltre a ambiente e memoria) lo HEAP (dedicato alle strutture dinamiche)

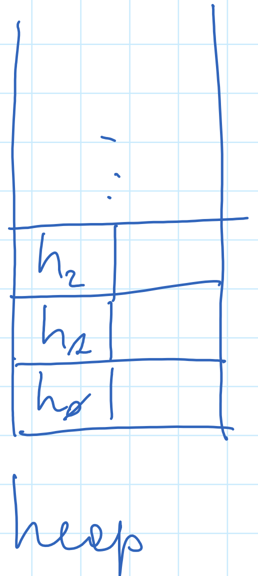
CONCRETAMENTE: si dedicano alla memoria a pila e alla memoria heap due zone diverse della memoria concreta del calcolatore, gestite in modo diverso.

lunedì 21 ottobre 2019 09:29

la memoria heap viene gestita come una sequenza di parole di memoria (senza struttura) in cui riservo spazio e cancello spazio esplicitamente con espressioni e comandi del programma.

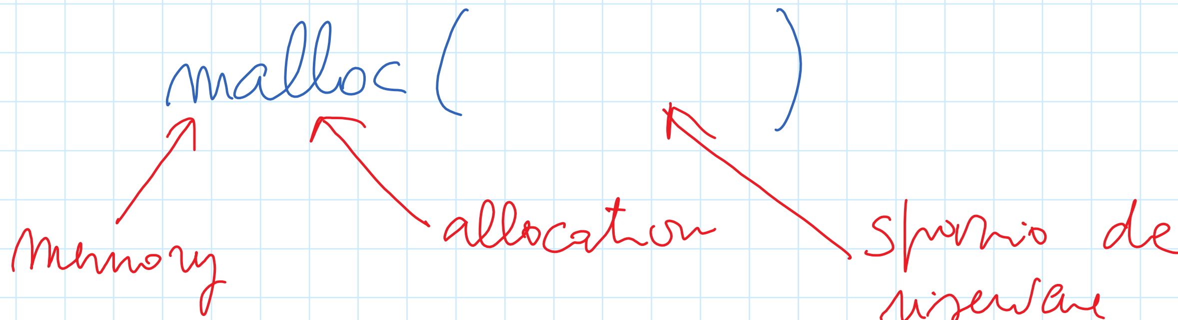
Allocazione e deallocazione sono esplicite.

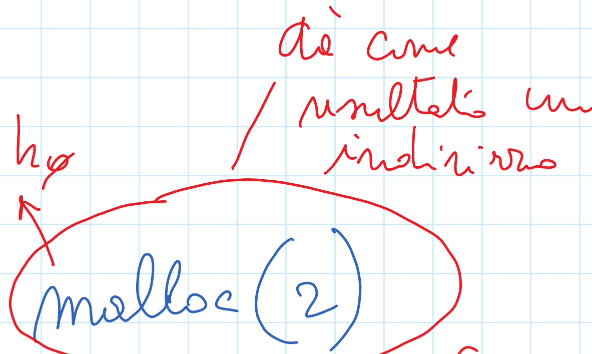
nuovo stato

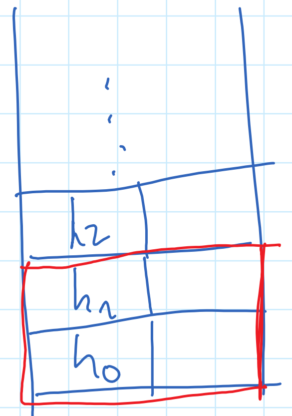


heap

le parole della memoria heap vengono riservate con l'espressione







l'effetto è quello di riservare due parole di memoria sullo heap e restituire l'indirizzo della prima parola riservata

primme parole riservate

Il C mette a disposizione una funzione

sizeof

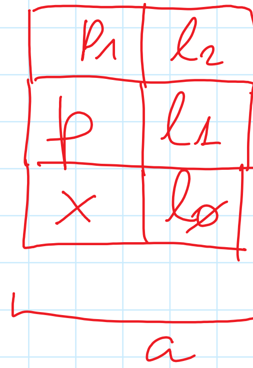
che dato un tipo restituisce il numero di parole di memoria che servono per memorizzare valori di quel tipo.

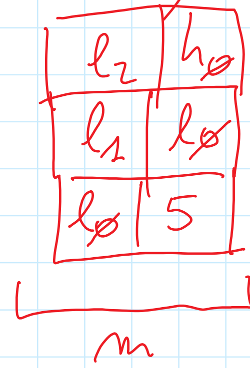
sizeof(int)

il risultato è il numero di parole che servono per memorizzare un intero (dipende dalla macchina)

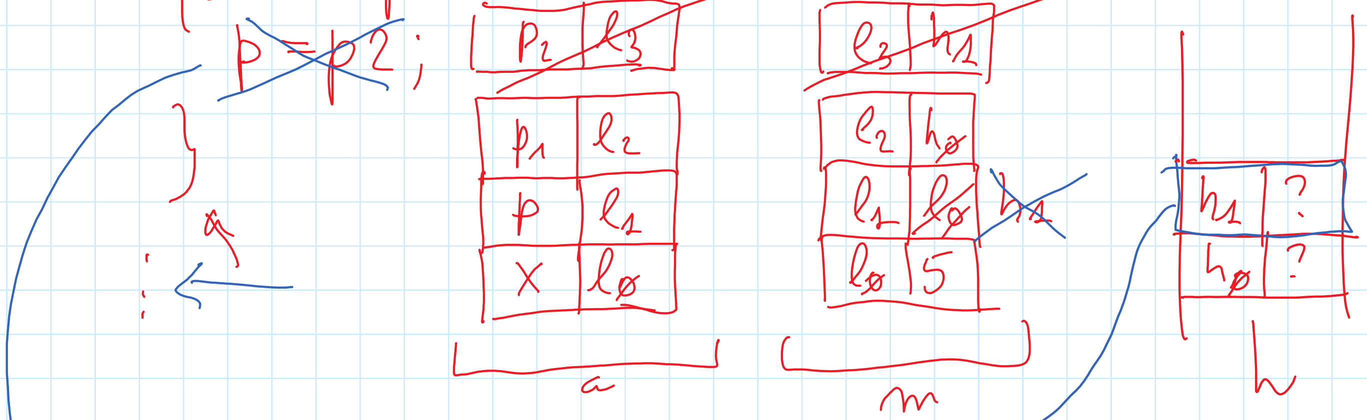
```
{ int x = 5;
  int * p = & x;
  int * p1 = malloc(sizeof(int));
```

$h_\emptyset$





```
{ int x = 5;
  int* p = &x;
  int* p1 = malloc(sizeof(int));
  { int* p2 = malloc(sizeof(int));
    p = p2;
  }
  ...
```



← Stanno sopravvive alle chiusure dei blocchi

Se non ci fosse stato questo assegnamento la locazione h_1 sarebbe rimasta riservata (allocata) ma non più accessibile. (GARBAGE)

Esistono dei programmi (GARBAGE COLLECTOR) che periodicamente "ripuliscono" la memoria heap.

In C non esiste un garbage collector, la memoria va deallocata esplicitamente con il

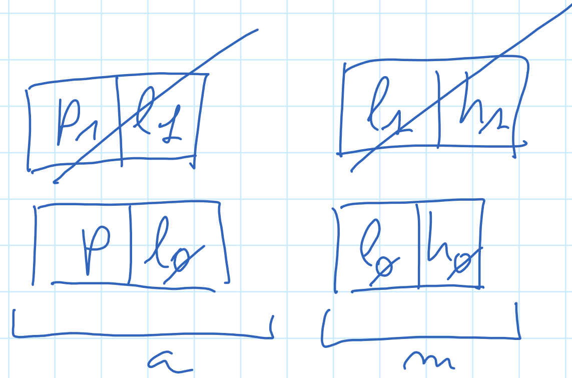
collector, la memoria va deallocata esplicitamente con il comando

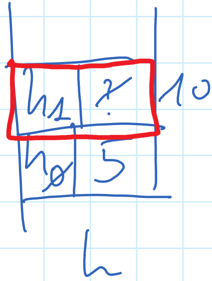
free (_);

variabile puntatore alla prima parola della memoria da riservare;

lo spazio di memoria da liberare viene dedotto dal tipo della variabile puntatore

```
{ int *p = malloc(sizeof(int));
  *p = 5;
  { int *p1 = malloc(sizeof(int));
    *p1 = 10;
    :
    free(p1);
  }
}
```





In C esiste la possibilità di dichiarare nuove strutture dati con il costrutto STRUCT che permette di costruire strutture composte da elementi di tipo diverso

ES:

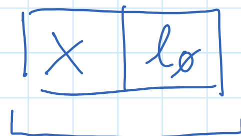
```c
struct el
{
    int a;
    int b;
    float c;
} x
```

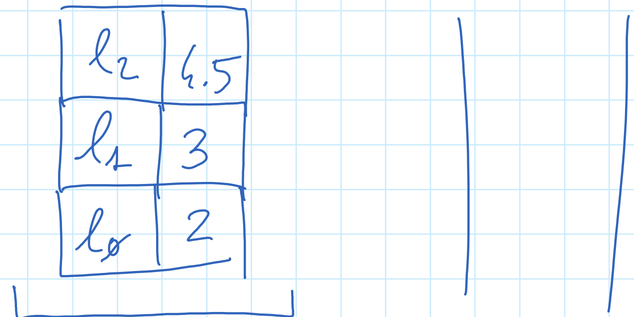
nome

nuovo tipo che ha nome struct el

è il tipo di struttura con 3 "campi", due interi e uno float con nomi a, b e c, rispettivamente

x è una variabile che ha tipo struct el





i campi si individuano con la notazione x.a, x.b e x.c

```c
x.a = 2;
x.b = 3;
```

[illegible]

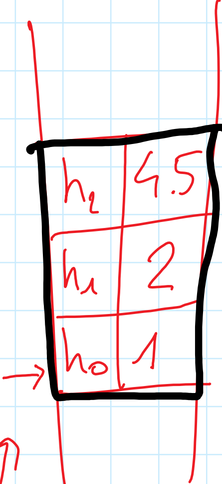
$x \cdot b = 3;$

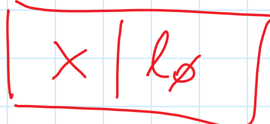
$x \cdot c = 4.5;$

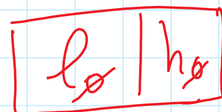
lunedì 21 ottobre 2019 10:22

```
struct el
{ int a;
  int b;
  float c;
} *X;

X = malloc(sizeof(struct el));
```







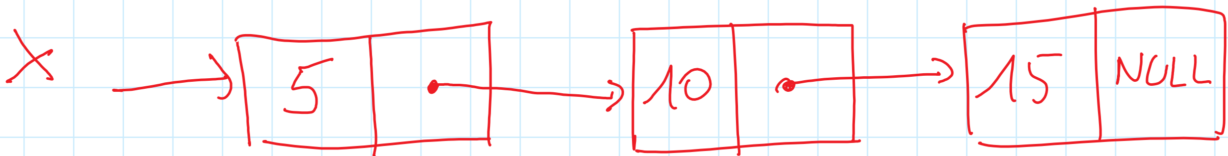

```
(*x).a = 1;
(*x).b = 2;
(*x).c = 4.5;
```

```
free(x);
```

lunedì 21 ottobre 2019 10:27

```
struct el
{ int info;
  struct el * next;
}
```

NULL



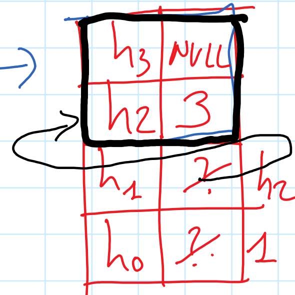
LISTA CONCATENATA (LINKED LIST)

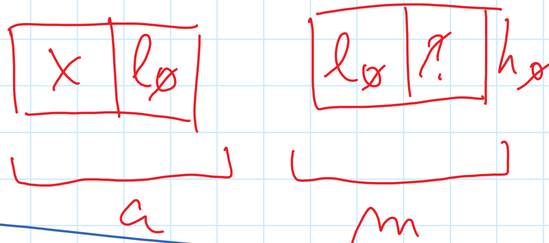
o

LISTA semplicemente

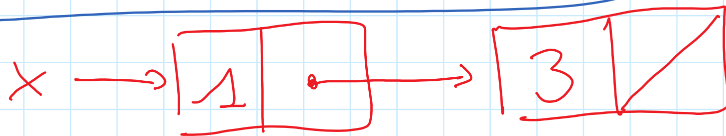
```
typedef
  struct el
    { int info;
      struct el * next;
    }
  ElementoDiListe;

ElementoDiListe * x;
x = malloc (sizeof (ElementoDiListe));
(*x).info = 1;
(*x).next = malloc (sizeof (ElementoDiListe));
```






```
(*(*x).next).info = 3;
(*(*x).next).next = NULL;
```



Notazione semplificata

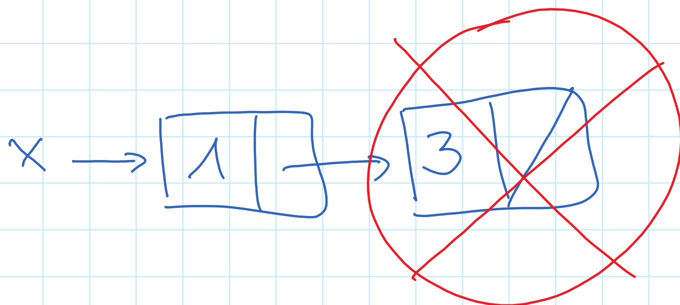
$(*x).info \equiv x \to info$

$(*x).info \equiv x \rightarrow info$

$(*x).next \equiv x \rightarrow next$

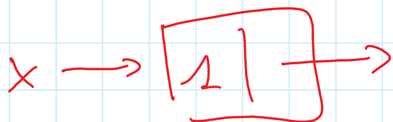
x → next → info = 3;

x → next → next = NULL

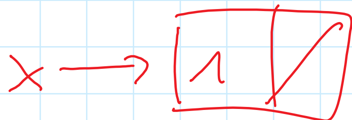


Voglio cancellarlo

free(x → next)



x → next = NULL;



Vogliamo scrivere una funzione C che crea una lista di n elementi (che contengono i valori 1, ..., n) e restituisce il puntatore al primo elemento.

n > 0

ElementoDiLista → EDL

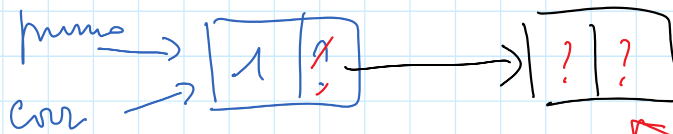
```
ElementoDiLista * crea (int n)
{ int i;
  ElementoDiLista * primo = malloc(sizeof(EDL));
```

primo → | ? | ? |

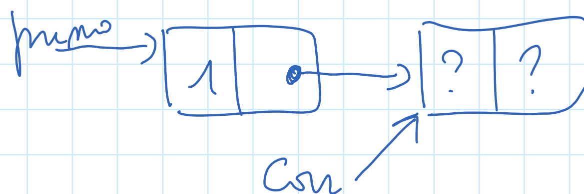
```
  ElementoDiLista * corr = primo;
```

primo, corr → | ? | ? |

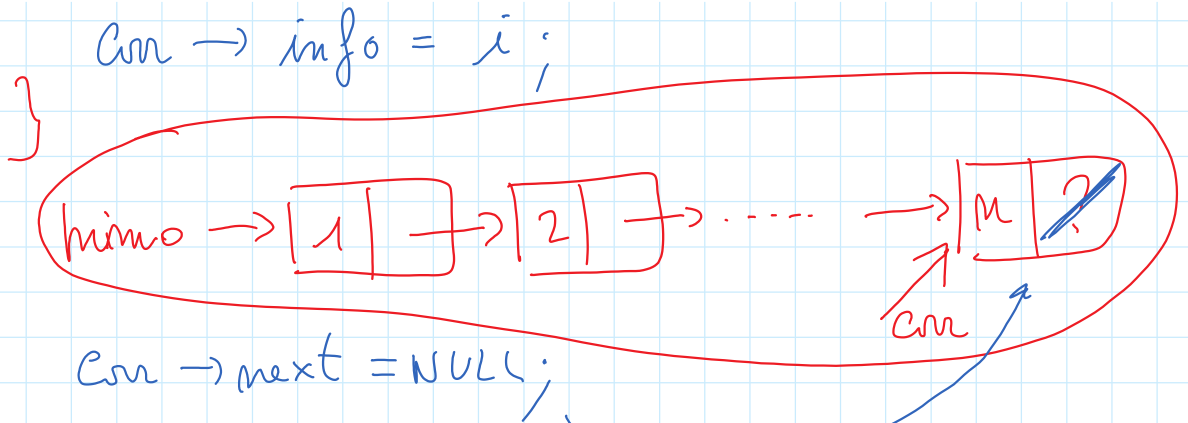
```
  corr -> info = 1;
```




```
  for (i=2; i<=n; i++)
  { corr->next = malloc(sizeof(EDL));
    corr = corr->next;
```



```
corr -> info = i;
}
```



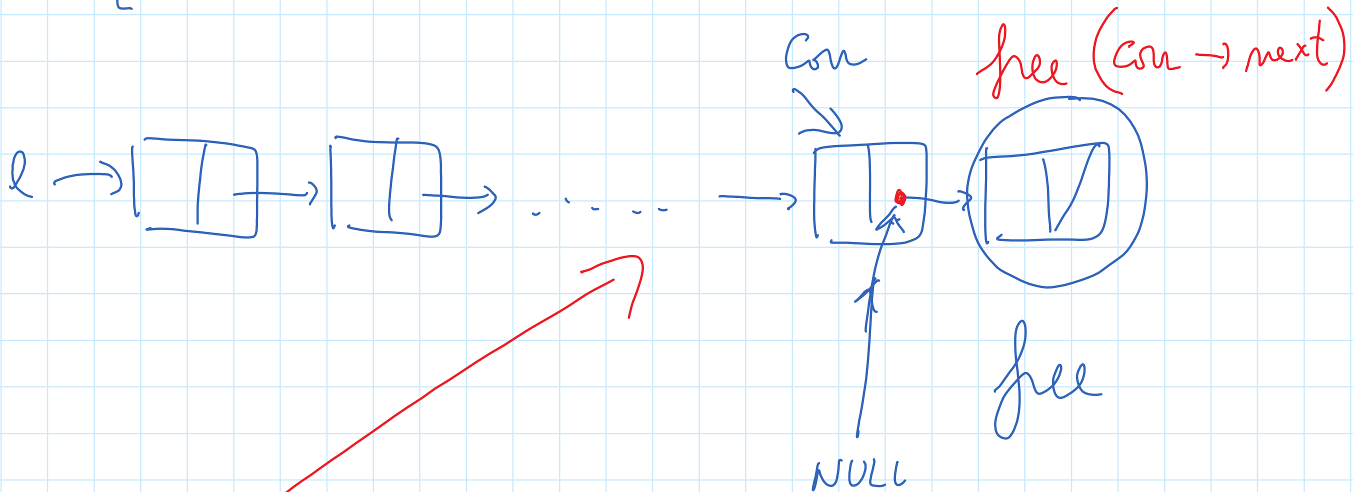

```
corr -> next = NULL;

return primo;
}

main()
{
    ElementoDiLista * l;
    l = crea(10);
    :
```

lunedì 21 ottobre 2019 10:56

```
ElementoDiLista* cancellaultimo(ElementoDiLista *l)
{ if (l==NULL) return l;
  else
  {
```



ATTENZIONE !

cor -> next = NULL;