Strutture dati in C

lunedì 6 novembre 2017 09:16

array

→ valori che hanno una struttura

- array è una struttura con valori omogenei (dello stesso tipo) con accesso mediante un indice.

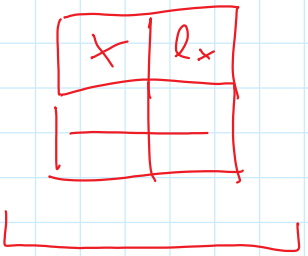
Un array è una struttura STATICA.
Quando si dichiara (crea) un array bisogna indicare la dimensione che non è modificabile

lunedì 6 novembre 2017 09:20

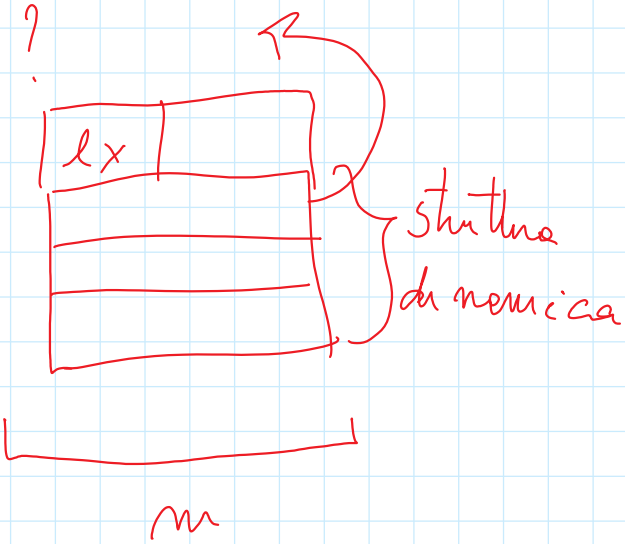
Esistono anche strutture DINAMICHE, per le quali è possibile modificare le dimensioni.

E' possibile memorizzare strutture dinamiche sulla memoria a pila?

E' difficile!!!



a



m

Si usa, per memorizzare strutture dinamiche, una memoria gestita in modo diverso dalla memoria a pila.

MEMORIA DINAMICA (heap)

lunedì 6 novembre 2017 09:24

La memoria dinamica non è gestita come una pila, le sue allocazioni non è gestita dalle esecuzioni dei blocchi (come la memoria a pila) ma è gestita esplicitamente dal programma.

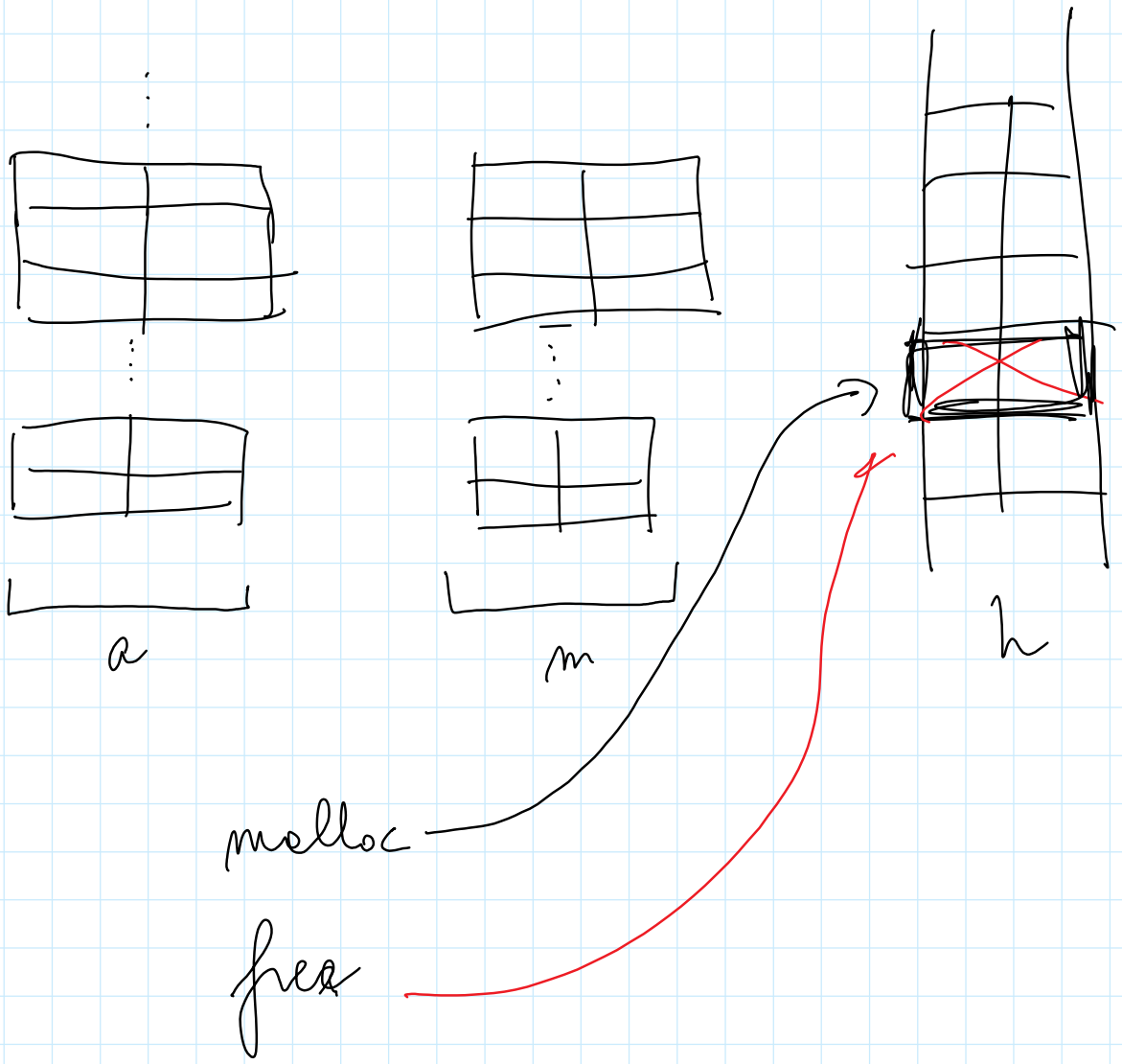
Esistono in C due operazioni che servono per creare spazio nella memoria dinamica (heap) e per cancellare spazio.

- malloc

memory allocation

- free

rendere disponibile (cancellare) memoria



La memoria heap viene gestita attraverso i suoi indirizzi.

- mette un valore nella memoria heap a un certo indirizzo,
- modifica il valore a un certo indirizzo.

Quindi, per agire nella memoria heap,

abbiamo bisogno di variabili intermedie.

MALLOC

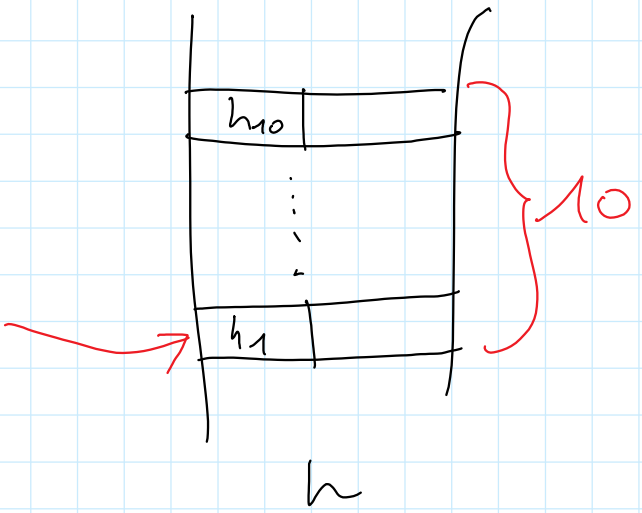
lunedì 6 novembre 2017 09:33

malloc riserva parole di memoria heap e restituisce l'indirizzo della prima parola del gruppo di parole riservate.

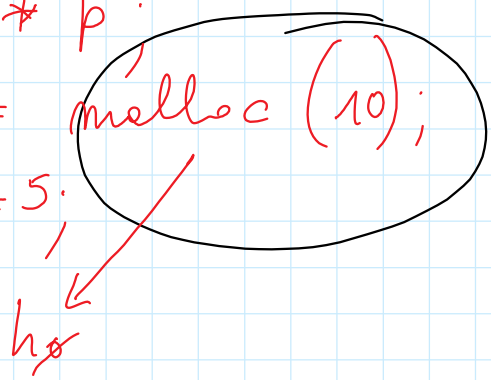
- malloc può riservare più di una parola di memoria.

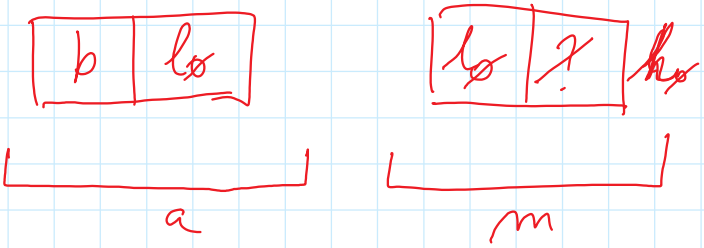
malloc (100) riserva (alloca) 100 parole di memoria e dimmi qual è il primo indirizzo di questo gruppo di parole

malloc (10)



```
{ int * p;
  p = malloc (10);
  *p = 5;
```





Le variabili puntatore sono puntatori, indifferentemente, alla memoria stack oppure alla memoria heap.

lunedì 6 novembre 2017 09:42

Come faccio a sapere quante memoria devo allocare per valori di un certo tipo.

int
float
double
⋮

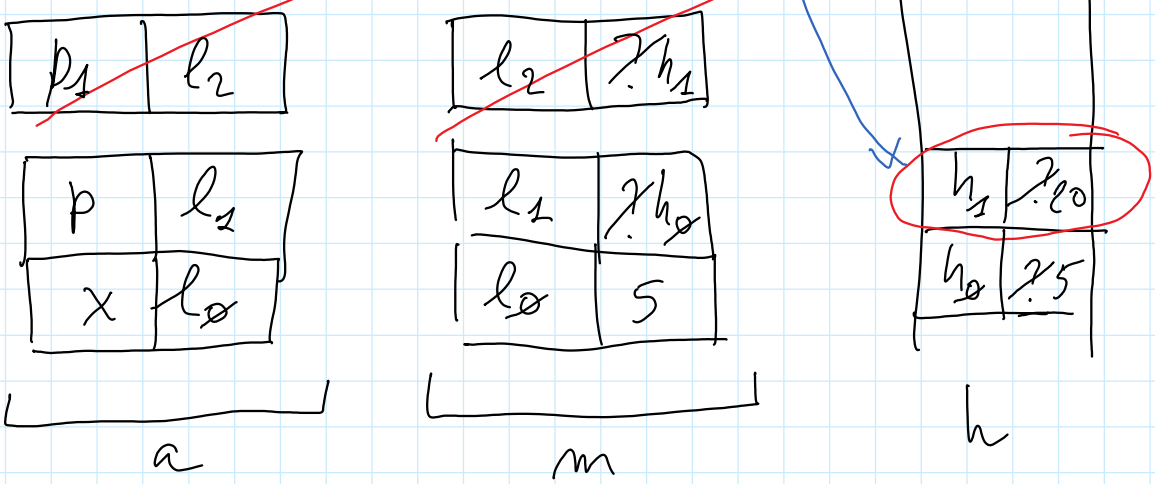
sizeof (tipo)

unità: il numero di parole di memoria che sono necessarie per memorizzare un valore di quel tipo

```
{ int x = 5;
  int * p;
  p = malloc ( sizeof (int));
  { int * p1;
    p1 = malloc (sizeof (int));
    *p = x;
    *p1 = 20;
  }
  ⋮
```

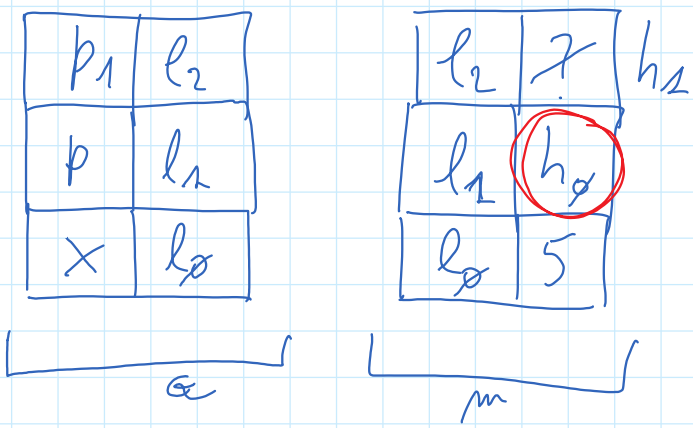
free (p1);

garbage



```
{ int x = 5;
  int *p = malloc(sizeof(int));
  int *p1;
  { int *p2 = malloc(sizeof(int));
    p1 = p2;
  }
  free(p);
  *p = 5;
  *p1 = 10;
```

DANGLING REFERENCE

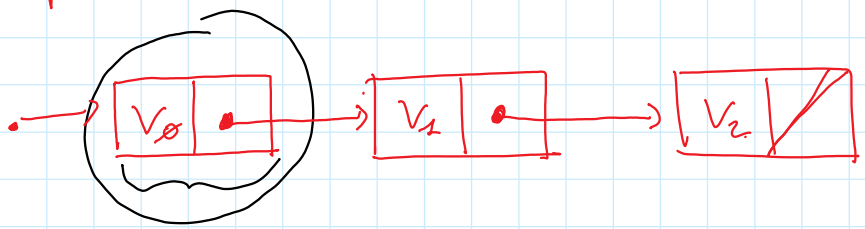


LISTA

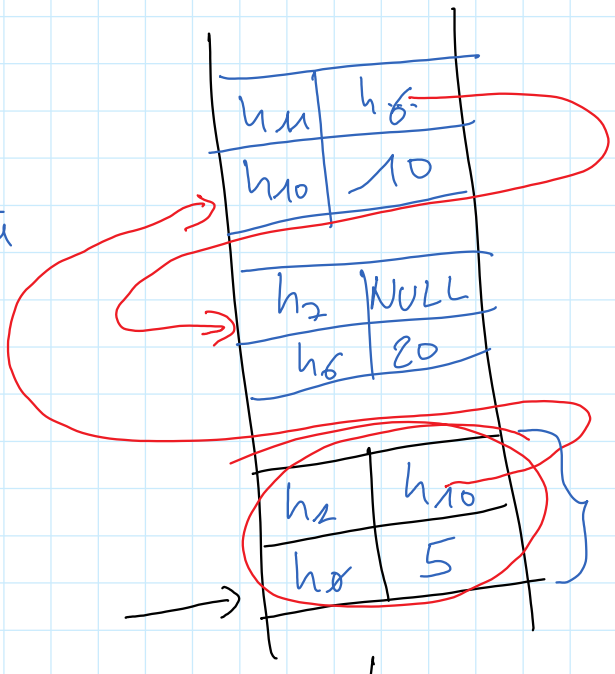
lunedì 6 novembre 2017 10:16

Lista è una struttura dati omogenea (con elementi dello stesso tipo) DINAMICA, cioè può crescere o diminuire.
Posso aggiungere elementi e togliere elementi.
L'accesso agli elementi di una lista è SEQUENZIALE. Cioè si accede al primo elemento, attraverso il primo elemento si può accedere al secondo, e così via.

Graficamente



NULL
puntatore a niente




5
10
20

lunedì 6 novembre 2017 10:26

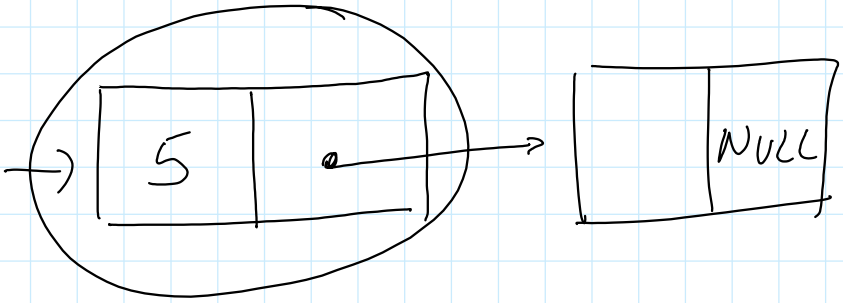
nome della struttura

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

il primo valore (campo) ha tipo int e si può individuare attraverso il nome info

struttura STRUCT è una struttura con oggetti di tipo diverso (in generale) individuati attraverso un nome.

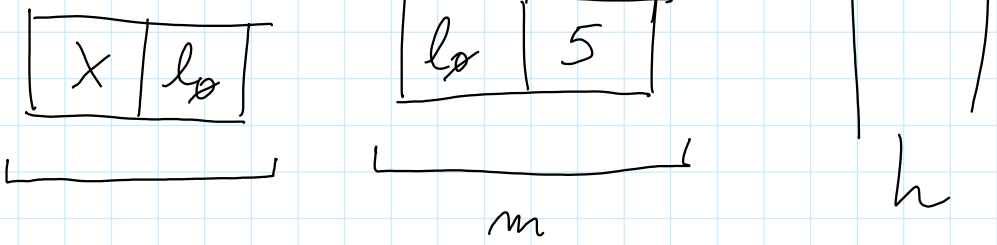
struct el



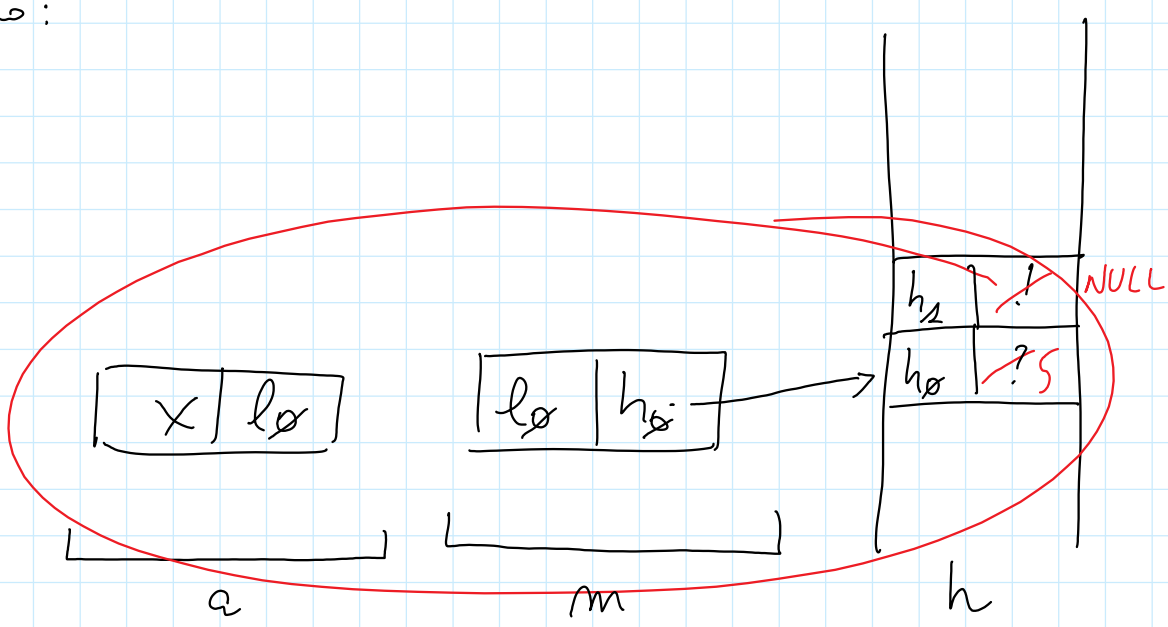
```
struct el
{ int info;
  struct el * next;
}

struct el x;
x.info = 5;
x.next = NULL;
x.info = x.info + 1;
```

il campo intero di x che ha nome info



Vorremo:




```
struct el * x;
```

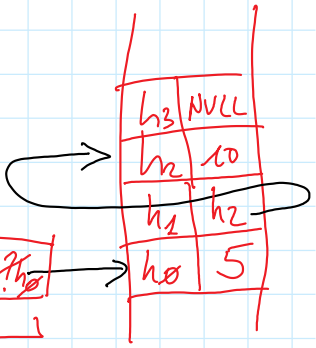
```
struct el * x;
x = malloc (sizeof (struct el));
```

~~x.info~~

```
x -> info = 5;
x -> next = NULL;
```

lunedì 6 novembre 2017 10:41

Vogliamo creare le liste:

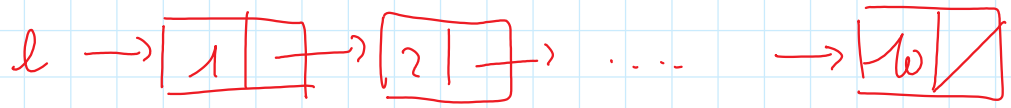



```
{ struct el { ... }
  struct el * x;
```

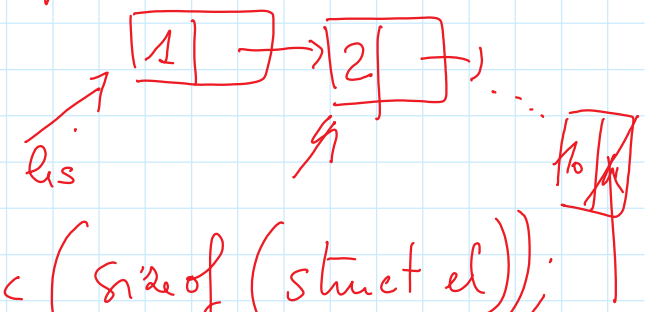

```
x = malloc(sizeof(struct el));
x -> info = 5;
x -> next = malloc(sizeof(struct el));
x -> next -> info = 10;
x -> next -> next = NULL;
```

Vogliamo creare la lista:

lunedì 6 novembre 2017 10:48




```
{ struct el {...}; int i; struct el *lis;
  struct el* l = malloc(sizeof(struct el));
  lis = l;
  l->info = 1;
  for (i=1; i<=9; i++)
    { l->next = malloc(sizeof(struct el));
      l->next->info = i+1;
      l = l->next;
    }
```



lis contiene il puntatore al primo elemento della lista creata.

```
l->next = NULL;
```