Titolo nota 09/11/2015

LISTE (in C)

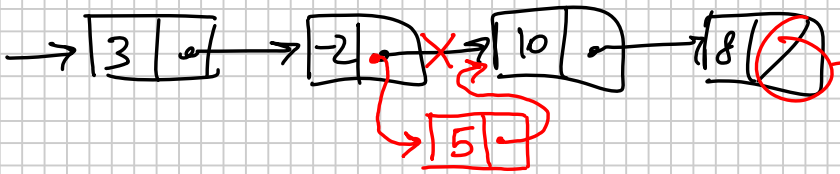
Una lista COLLEGATA (linked list) è una sequenza di oggetti dello stesso tipo (come gli array) ma di dimensione variabile. L'accesso a ciascun elemento NON è DIRETTO come nel caso degli array, ma è SEQUENZIALE

- mentre negli array l'accesso ad un elemento avviene direttamente attraverso l'indice dell'elemento `a[i]`
- nel caso delle liste l'accesso all'i-esimo elemento avviene scandendo tutti gli elementi precedenti.

l'accesso avviene sempre attraverso un COLLEGAMENTO al primo elemento

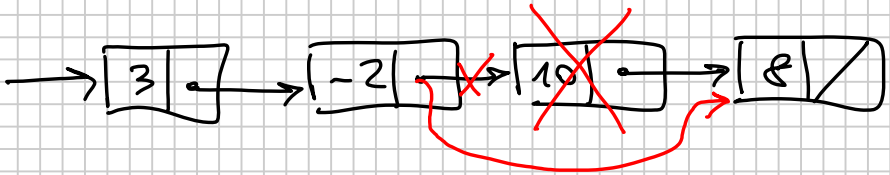
La lista è una struttura DINAMICA

- possiamo aggiungere o eliminare elementi in una lista



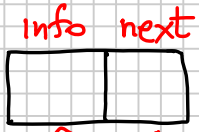
indica l'assenza di un elemento successivo (l'elemento è l'ultimo della lista)

AGGIUNTA di un ELEMENTO



CANCELLAZIONE di UN ELEMENTO

Come si realizzano le liste in C?
Per semplicità LISTE di INTERI



Dobbiamo rappresentare gli elementi di una lista

c'è un intero

un "collegamento" al prossimo elemento della lista (o il "collegamento" speciale per rappresentare l'ultimo elemento)

STRUTTURE in C

el

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

el è una struttura fatta da 2 componenti:
- la prima, denominata info, è di tipo int
- la seconda, " next, è un puntatore ad una struttura dello stesso tipo

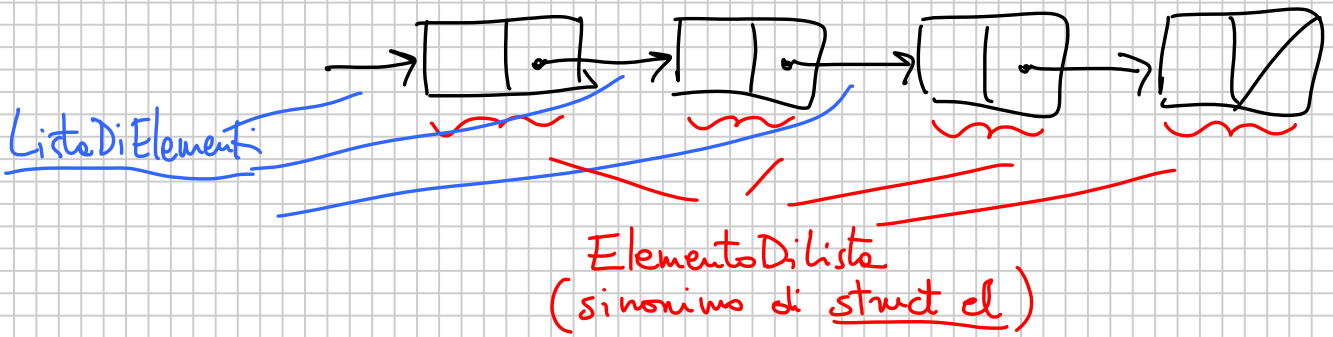
Per comodità, possiamo dare un NOME al nuovo tipo struct el che abbiamo definito

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

nuovo tipo definito sopra — nome per il nuovo tipo

```
typedef ElementoDiLista *ListaDiElementi;
```

puntatore ad un elemento di lista — nome del tipo "puntatore" ad un elemento di lista



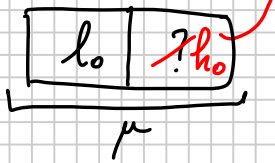
```
struct el {int info; struct el *next;}
typedef struct el ElementoDiLista;
typedef ElementoDiLista *ListaDiElementi;

ListaDiElementi lista;
```

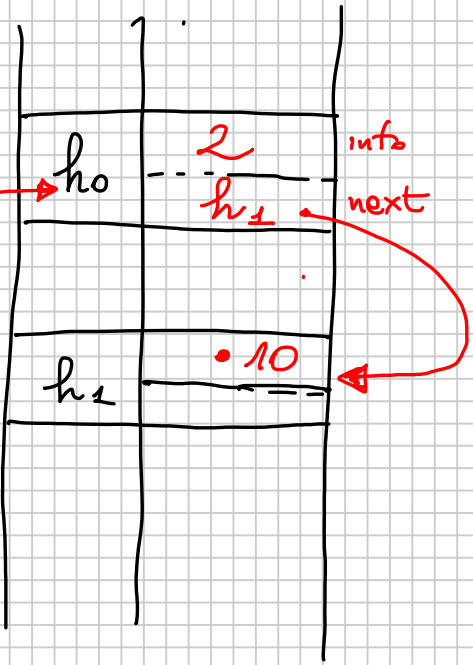
è un tipo come tutti gli altri, che può essere attribuito ad una variabile

```
lista = malloc (sizeof (ElementoDiLista));
/* come si accede alle componenti della struttura? */
lista -> info = 2;
```

è la componente info della struttura "puntata" dalla variabile lista

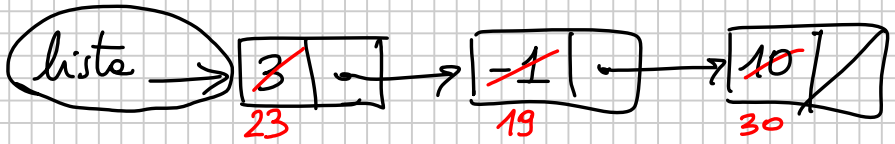
```
lista -> next = malloc (sizeof (ElementoDiLista));
(lista -> next) -> info = 10;   lista -> next -> next = NULL;
```





costante predefinita in <stdlib.h> che rappresenta il puntatore "vuoto"

Procedura che, data una lista di interi, aumenta di n tutti gli elementi della lista

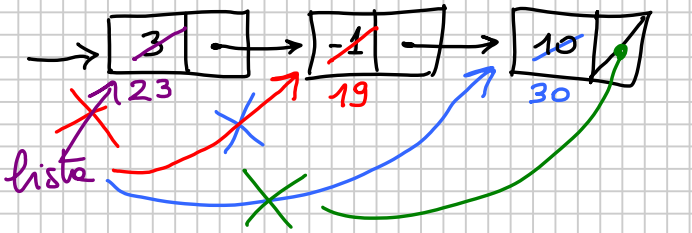


n = 20

```
void aumenta(ListaDiElementi lista, int n)
{
    while(lista != NULL)
    {
        lista->info = (lista->info) + n;
        lista = lista->next;
    }
}
```

lista->next != NULL
no!! non modifica l'ultimo elemento


```
void aumenta(int a[], int dim, int n)
{
    int i;
    i = 0;
    while(i < dim)
    {
        a[i] = a[i] + n;
        i = i + 1;
    }
}
```



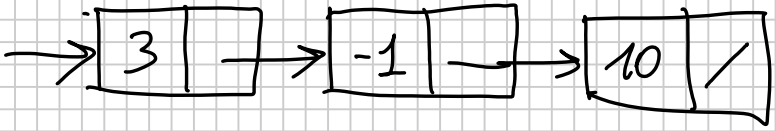
Scriviamo una funzione che, data una lista, ci dice quanto è lunga

```
int length (ListaDiInteri lista)
{ int lunghezza = 0;
  while (lista != NULL)
    { lunghezza = lunghezza + 1;
      lista = lista -> next;
    }
  return (lunghezza);
}
```

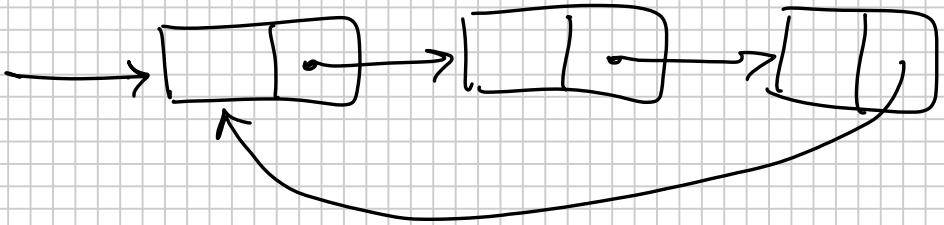
La lista è una STRUTTURA DATI RICORSIVA - Definizione induttiva

1 - la lista vuota (priva di elementi) è una lista (NULL è la lista vuota)

2 - Se ℓ è una lista allora  ℓ è una lista



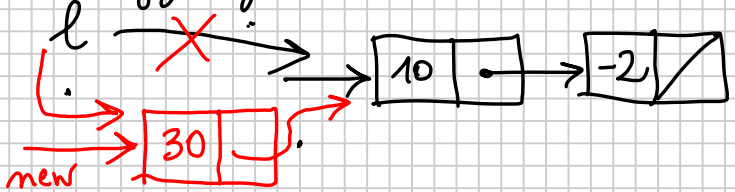
LISTE CIRCOLARI



DOUBLE LINKED LISTS

```
int *p = NULL;          non rappresenta la lista vuota !!
ListaDiElementi l = NULL;     questo sì !!
int x;
*p = 10;          errore perché p = NULL
p = &x;
*p = 10;          ok.
```

Vogliamo aggiungere un elemento x in cima ad una data lista l



aggiungiamo 30 in cima

```
void addT (ListaDiElementi l, int x)
{ ListaDiElementi new = malloc (sizeof (ElementoDiLista));
  new → info = x;
  new → next = l;
  l = new;
}
```

questa modifica viene fatta sul parametro della procedura non sulla lista del "chiamante"

non funziona per lo stesso motivo per cui non funziona questa procedura

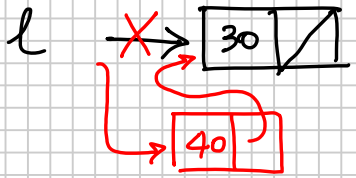
```
void incr (int x)
{ x = x+1; }
```

⇒

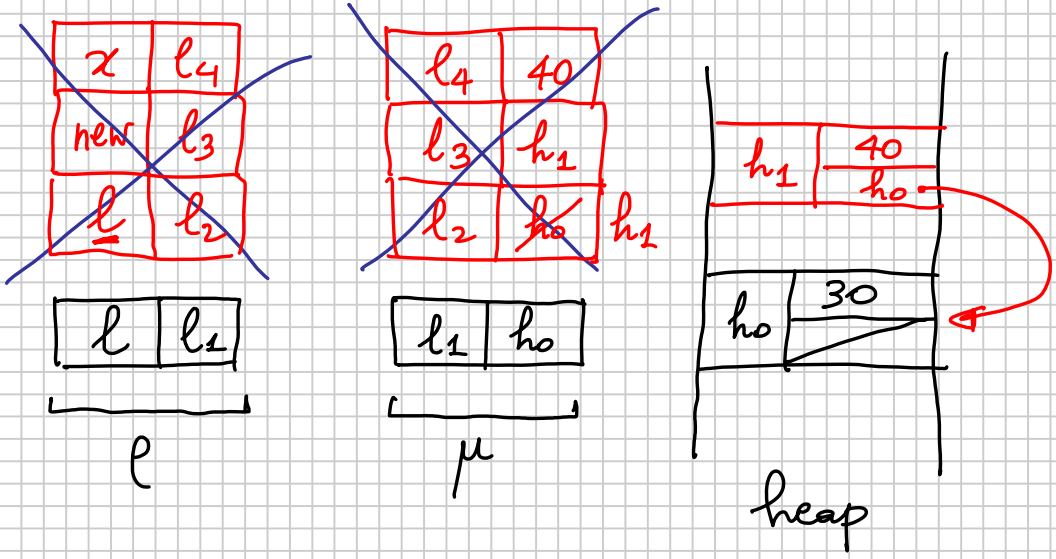
```
void incr (int *x)
{ *x = *x + 1; }
```

⇐

```
main ()
{ ListaDiElementi l;
  l = malloc(sizeof(ElementoDiLista));
  l → info = 30;
  l → next = NULL;
  addT (l, 40);
```




```
void addT (ListaDiElementi *l, int x)
  { ListaDiElementi new= malloc(...);
    new → info = x;
    new → next = *l;
   *l = new;
  }
```



```
main()
{ ListaDiElementi lista;
  lista = malloc(....);
  lista -> info = 30;
  lista -> next = NULL;
  addT(&lista, 40);
```

```
void addT (ListaDiElementi *l, int x)
  { ListaDiElementi new = malloc (....);
    new -> info = x;
    new -> next = *l;
    *l = new;
  }
```

Sem $*\mu \; \ell \; \mu =$

$\mu \; (\mu \; (\ell \; (\mu)))$