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

```
typedef struct el ElementoDiLista
```

struct el: definizione del tipo

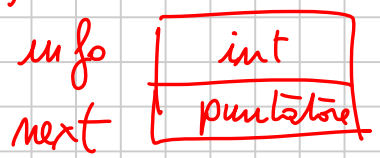
ElementoDiLista: nome del tipo

```
typedef ElementoDiLista * ListaDiElementi
```

ElementoDiLista *: definiz. del tipo

ListaDiElementi: nome del tipo

oggetto di tipo ElementoDiLista



oggetto di tipo ListaDiElementi

Titolo nota 10/11/2015

malloc è una funzione che "alloca" memoria heap e restituisce il puntatore (l'indirizzo) alla memoria "allocata"

malloc ( )

malloc (sizeof (ElementoDiLista))
malloc "alloca" la memoria per contenere uno struct el

8 bit

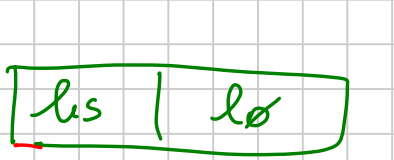
argomento il numero di parole di memoria (byte) da "allocare"

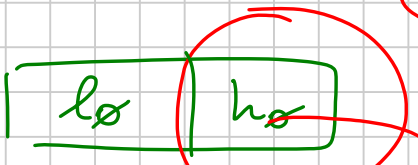
sizeof (tipo) dice quanta memoria serve per oggetti di questo tipo

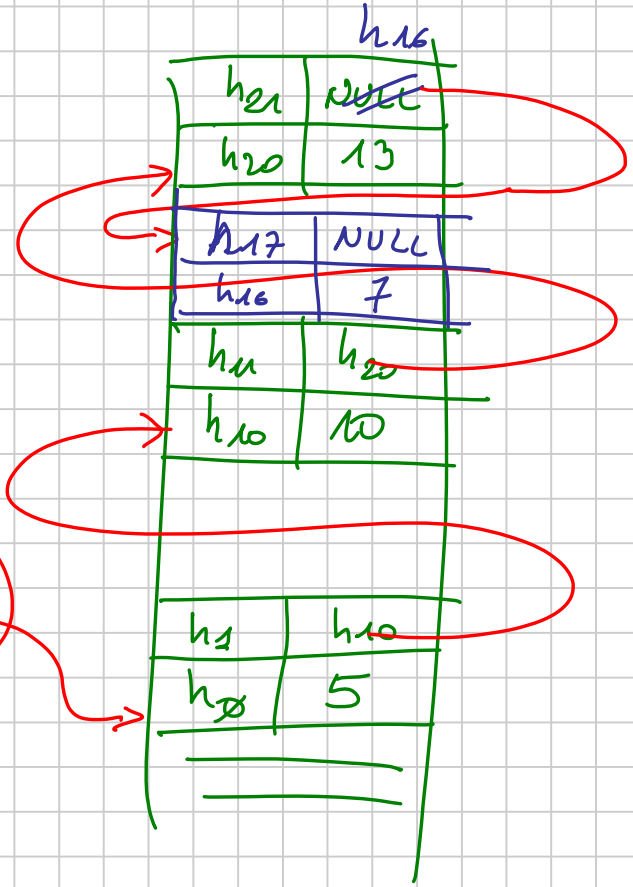
Titolo nota 10/11/2015

Date una lista, abbiamo vista una procedura che aggiunge un elemento in coda a una lista non vuota

```
void addLe (ListaDiElementi l, int el)
{
  ;
  .
}
```





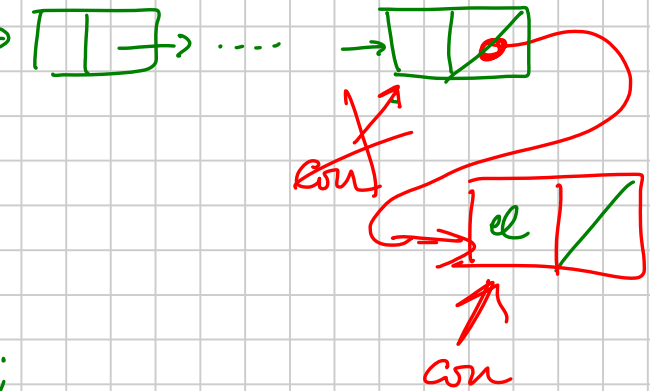


Procedura che aggiunge un elemento in coda a una lista non vuota

Titolo nota 10/11/2015

```
void addc ( ListeDiElementi l, int el)
{ ListeDiElementi corr = l;
  while (corr -> next != NULL)
        corr = corr -> next;
  corr->next = malloc ( sizeof (ElementoDiListe ));
  corr -> next -> info = el;
  corr -> next -> next = NULL;
}
```

```
corr = corr -> next;
corr -> info = el;
corr -> next = NULL;
```



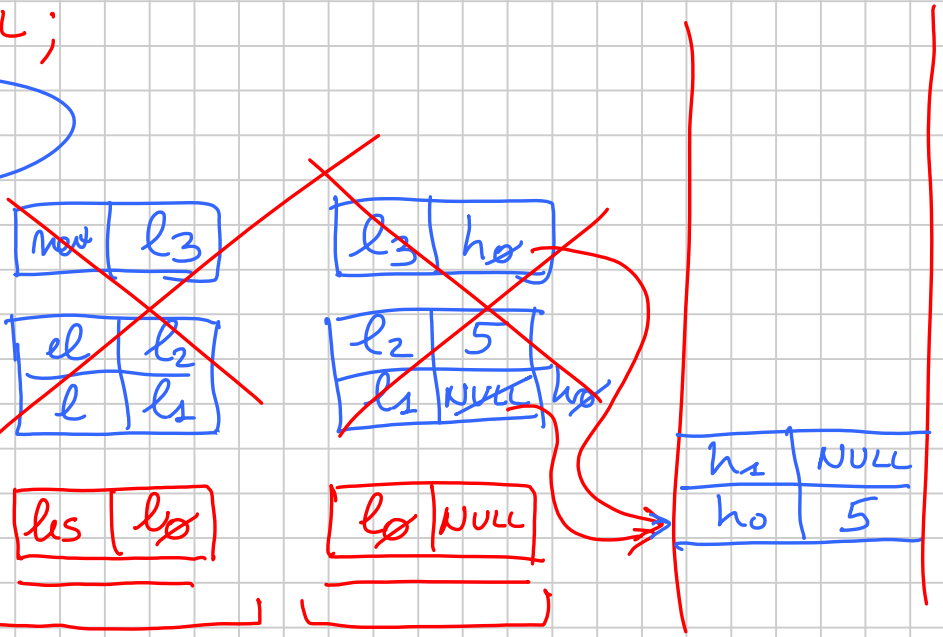
Titolo nota 10/11/2015

Se le liste può essere vuote:

```
void addLe (ListaDiElementi l, int el)
{
    ListeDiElementi new = malloc(sizeof(EDL));
    new -> info = el; new -> next = NULL;
    if ( l == NULL)  l = new;
    ...
}

main()
{
    ListeDiElementi lis = NULL;
    addLe (lis, 5);
}
```

NULL 5

l = ~~NULL~~ → new → | el | /



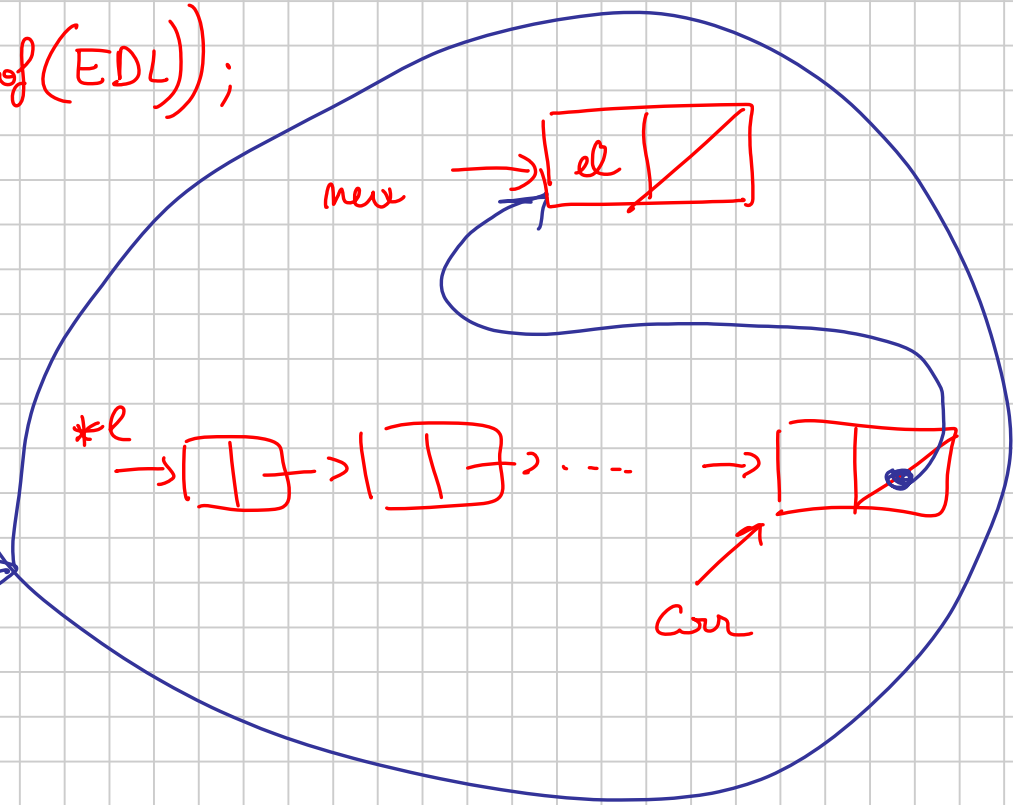
Titolo nota

10/11/2015

Se le liste può essere vuote:

Titolo nota 10/11/2015

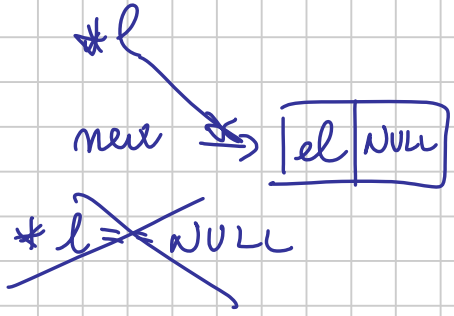
```
void addc (ListaDiElementi *l, int el)
{
    ListeDiElementi new = malloc(sizeof(EDL));
    new -> info = el; new -> next = NULL;
    if (*l == NULL) *l = new;
    else { ListaDiElementi corr = *l;
           while (corr -> next != NULL)
               corr = corr -> next;

           corr -> next = new;
         }
}
```

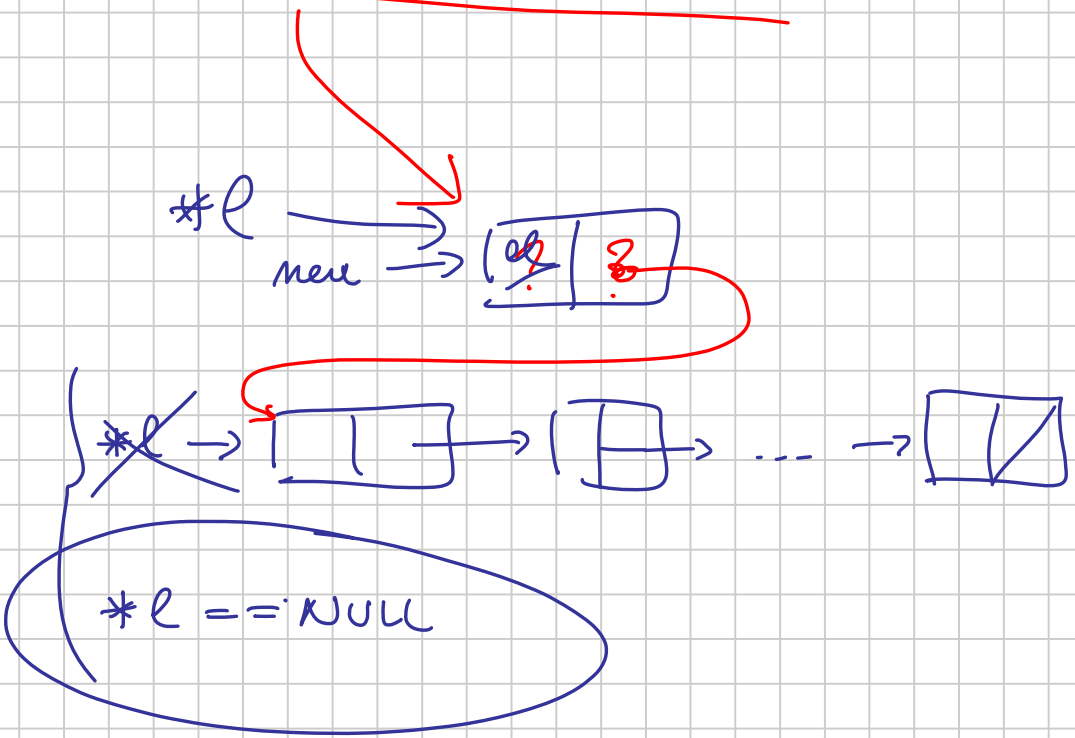


Aggiungere un elemento in testa a una lista (anche vuota)

Titolo nota 10/11/2015

```
void addIt (ListaDiElementi * l, int el)
{  ListaDiElementi new = malloc (sizeof (ElementoDiLista));
   new -> next = *l;
   new -> info = el;
   *l = new;
}
```

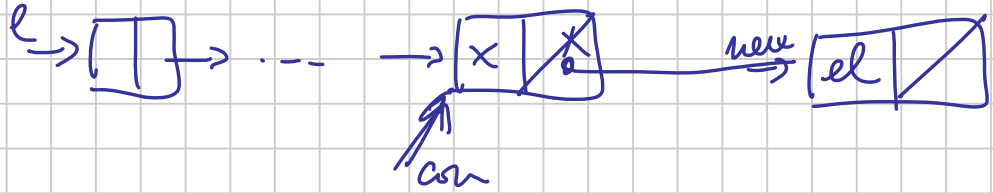


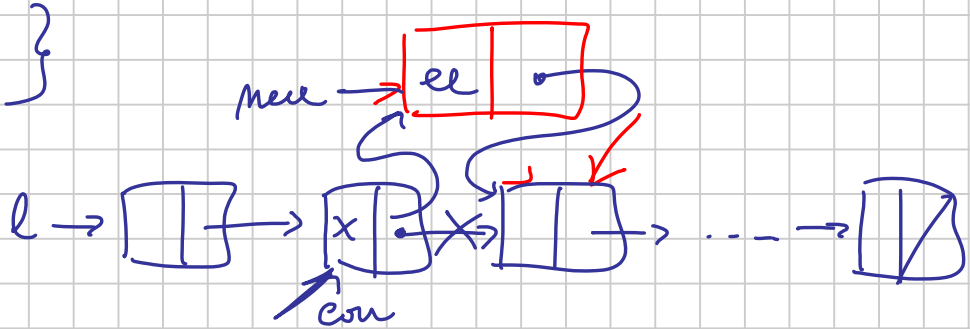


Titolo nota 10/11/2015

Aggiungere un elemento, el, dopo la prima occorrenza di un valore x.
Se x non compare non facciamo niente.

```
void addDopoX (ListaDiElementi l, int el, int x)
{ ListaDiElementi corr = l;
  int trovato = 0;
  while (corr != NULL && !trovato)
    if (corr -> info == x) trovato = 1;
      else corr = corr -> next;
  if (trovato)
  {ListaDiElementi new = malloc (sizeof(EDL));
   new -> info = el;
   new -> next = corr -> next;
   corr -> next = new;
  }
}
```

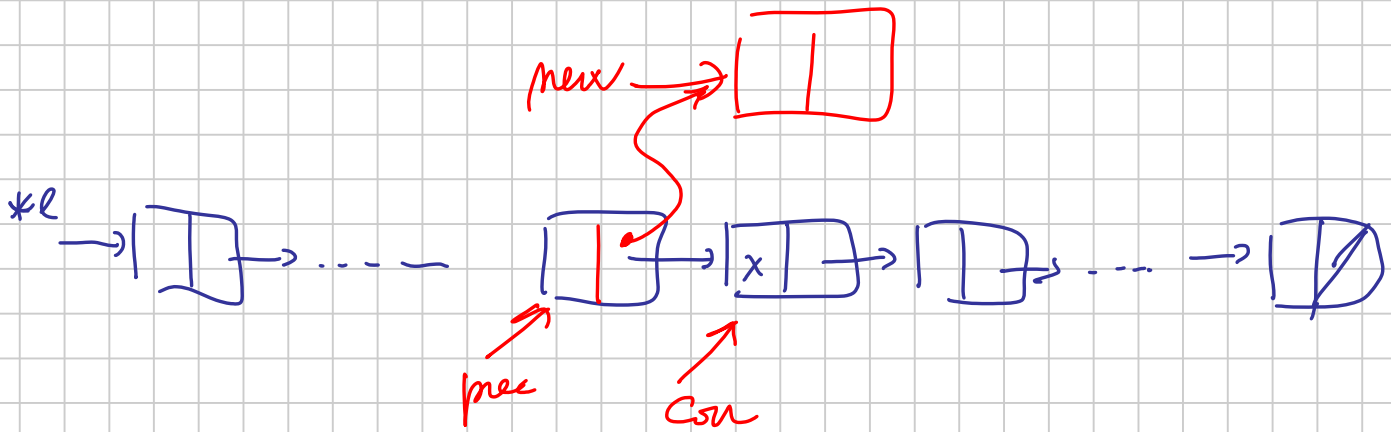




Titolo nota 10/11/2015

Aggiungere un elemento, el, prima della prima occorrenza di x.
Se x non compare la lista rimane immutata.

```
void aoldprimex(ListeDiElementi *l, int el, int x)
```



Titolo nota 10/11/2015

Aggiungere un elemento, el, prima della prima occorrenza di x.
Se x non compare la lista rimane immutata.

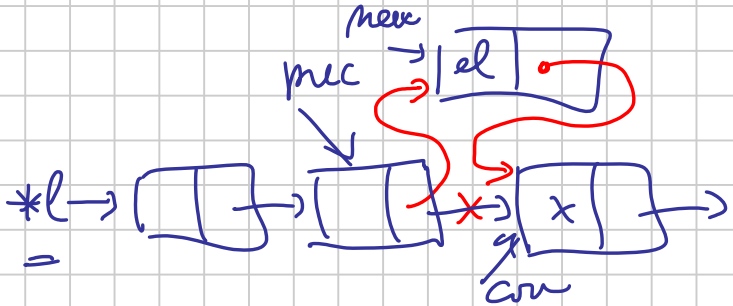
```
void aoldprimex(ListaDiElementi *l, int el, int x)
{ ListaDiElementi corr = *l; ListaDiElementi prec = NULL;
  int trovato = 0;
  while (corr != NULL && !trovato)
    if (corr->info == x) trovato = 1;
      else { prec = corr;
             corr = corr->next;
           }
```

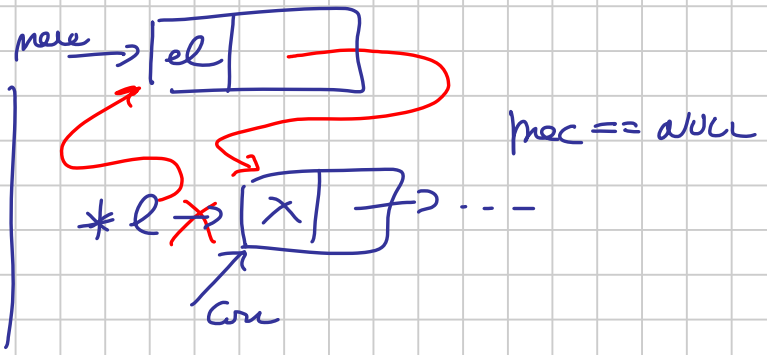
~~prec = prec->next;~~
~~corr = corr->next;~~

↓ vedi pagina nuova

Titolo nota 10/11/2015

```
if ( trovato )
{ ListaDiElementi new = malloc (sizeof (ElementoDiLista));
  new -> info = el; new -> next = cor;
  if ( prec == NULL ) { new -> next = *l;
                        *l = new;
                      }
  else { new -> next = cor;
         prec -> next = new;
       }
}
```

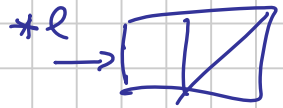


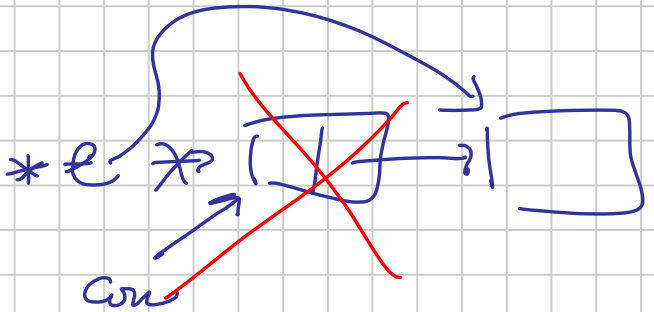


Cancellare l'elemento in testa (se la lista non è vuota)

Titolo nota 10/11/2015

```
void canct (ListeDiElement * l)
{ if (*l != NULL)
    { ListeDiElement corr = *l;
      *l = *l -> next;
      free (corr);
    }
}
```





Cancellare l'ultimo elemento da una lista (se c'è)

Titolo nota 10/11/2015

```
void cancc (ListeDiElementi *l)
{ if (*l != NULL)
  { if (*l->next == NULL) { free(*l); *l = NULL; }
    else { ListeDiElementi prec = *l;
           ListeDiElementi corr = *l->next;
           while (corr->next != NULL)
             { prec = corr;   <=≡   prec = prec->next;
               corr = corr->next;
             }
           free(corr);
           prec->next = NULL;
    } } }
```