STRUTTURE DATI DINAMICHE

Esempi già visti di uso dei puntatori:
int a; int *p = &a, *q;
..
q=p; p=NULL; if (p==q) o (p==NULL) ...
(*p) =3;

- dinamiche:
  - a struttura nota a compile-time (tipo);
  - creazione e deallocazione gestite dal programmatore;
  - referenza solo tramite indirizzo (puntatore) perché non hanno nome.

Creazione e distruzione variabili dinamiche

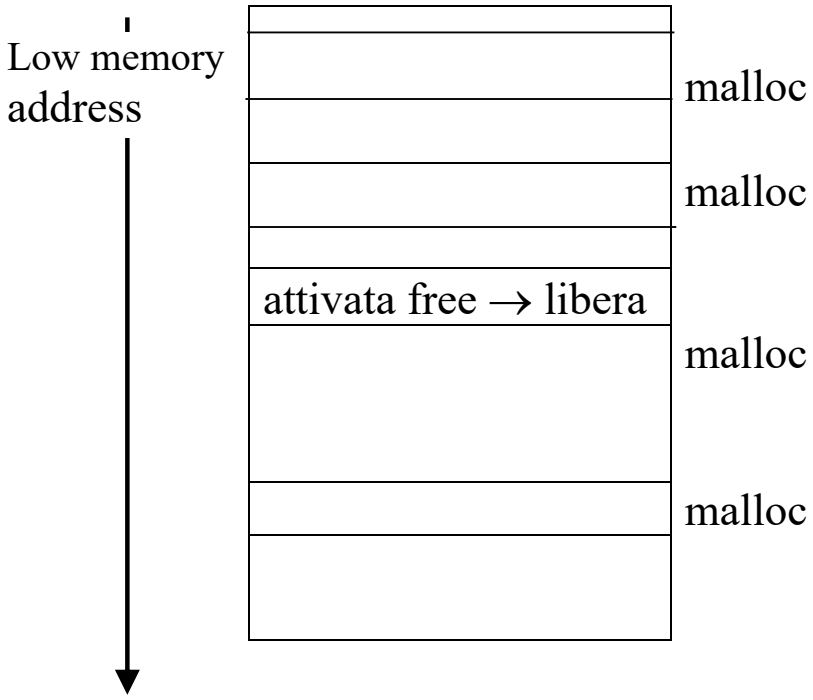
- import dal modulo di libreria #include <stdlib.h>

- void malloc(int num);

- alloca num bytes e ritorna il puntatore (NULL ≡ problemi);
- il risultato void e il casting:
  es. p = (rec *) malloc(sizeof(rec));
- l'allocazione avviene nell'area di Heap - heap overflow

void free(void *pointer);
  es. free(p);
- dealloca spazio occupato dalla variabile
- pointer non più utile ma con indirizzo

Allocazione e deallocazione nell'area di HEAP



Attenzione

- allocazione di blocchi "potenzialmente" di dimensione variabile

  ⇓

  spazio heap libero, ma frammentato blocchi liberi troppo piccoli

- Non ci sono meccanismi automatici di gestione dell'area di heap

Allocazione vettore di dimensione dinamica?

```
int main()
{int quanti, i, *p;
  scanf("%d",&quanti);
  p=malloc(quanti*sizeof(int));          if (p==NULL) error
….
  for (i=0; i<quanti;i++) p[i]=i; oppure *(p+i)=i;

  for (i=0; i<quanti;i++) printf(" %d", p[i]); oppure  *(p+i));
```

- Dopo malloc il vettore è di dimensioni fissate
- Puntatore p svolge il ruolo del nome del vettore

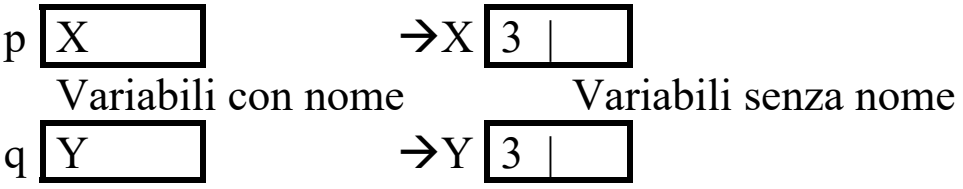
Osservazione

char vet[8]={...}; vettore modificabile
int *p; p=malloc(sizeof(char)*8); vettore modificabile

char *p = “def”;

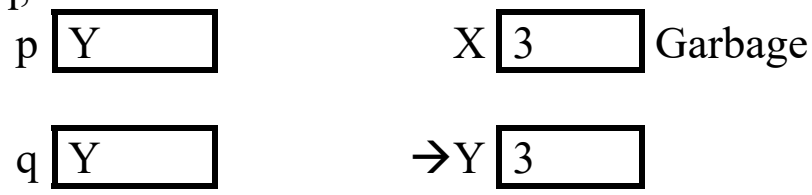
Gestione memoria e problemi

```
p = (rec *) malloc(sizeof(rec));          if (p==NULL) error
q = (rec *) malloc(sizeof(rec));          if (q==NULL) error
(*p).a =3; p->a=3;
(*q).a =3; q->a=3;
```



a) produzione diretta di garbage: variabile dinamica irraggiungibile.

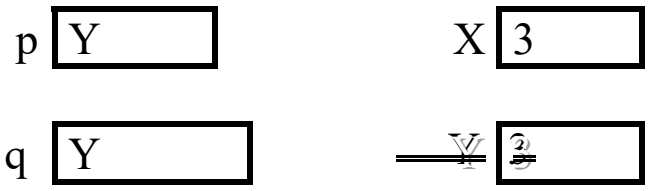
p=q;



(non esiste garbage collector)

b) generazione dangling reference: puntatore con indirizzo non valido

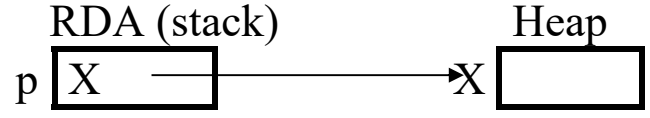
free(q);



p->a = 2;
q->a = 2; errore in esecuzione

c) produzione indiretta di garbage:

```
void P()
{ int *p;   p = (int *) malloc(sizeof(int)); }

int main() {P();}
```



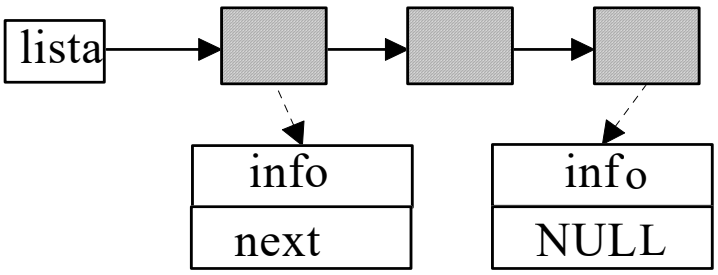
d) produzione indiretta di dangling reference:

```
#....
int *p;
void F()  { int n; p=&n;}

int main() { F();}
```

Strutture dati concatenate – tipo ricorsivo

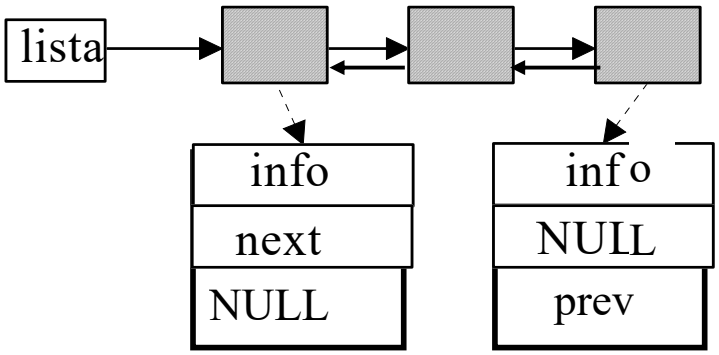
- insieme di elementi di tipo omogeneo
- collegamento tramite puntatori
- almeno un "handle" per accedere alla struttura

Lista monodirezionale

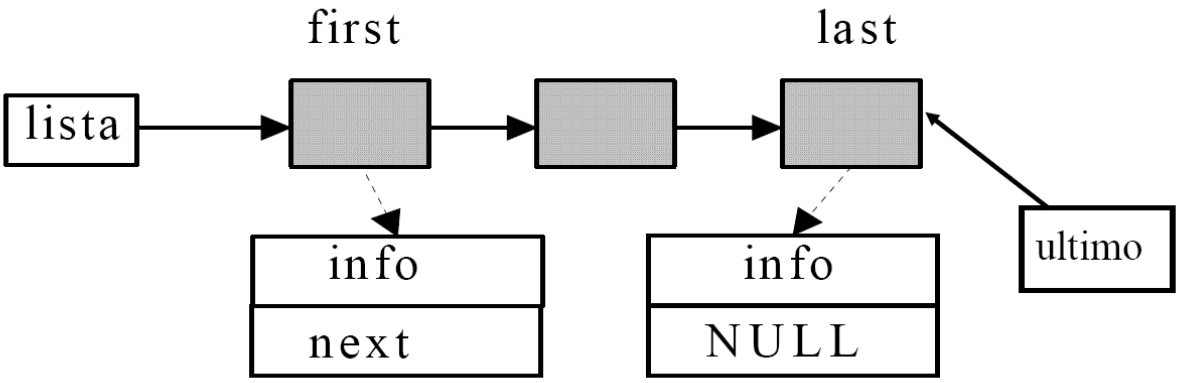


definizioni
struct el {int info; struct el *next;};
struct el *lista=NULL;

Lista bidirezionale
definizioni
struct el {int info; struct el *prev; struct el *next;};
struct el *lista=NULL;

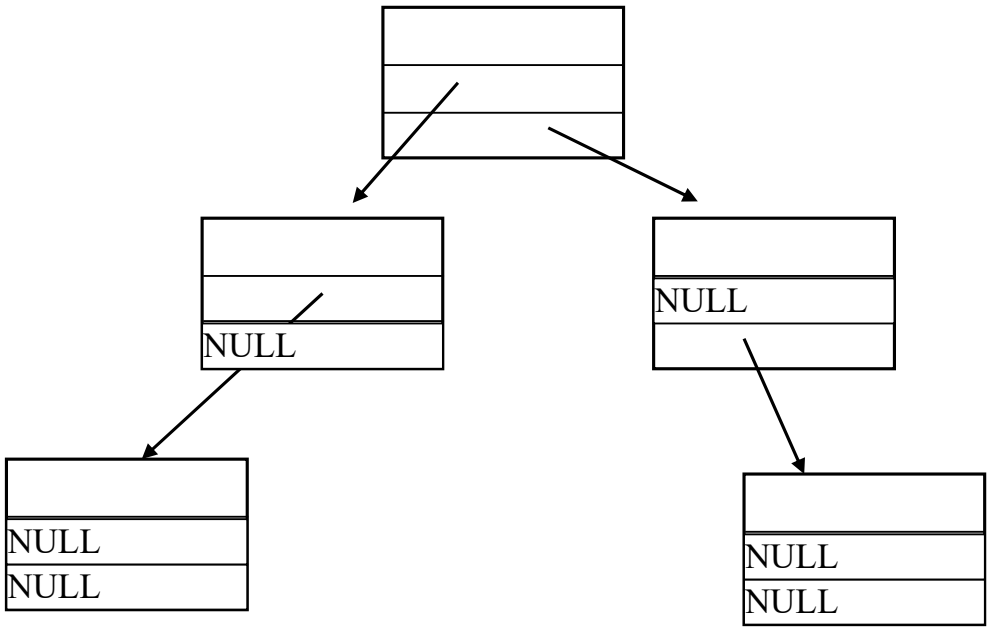


Lista monodirezionale con doppio handle

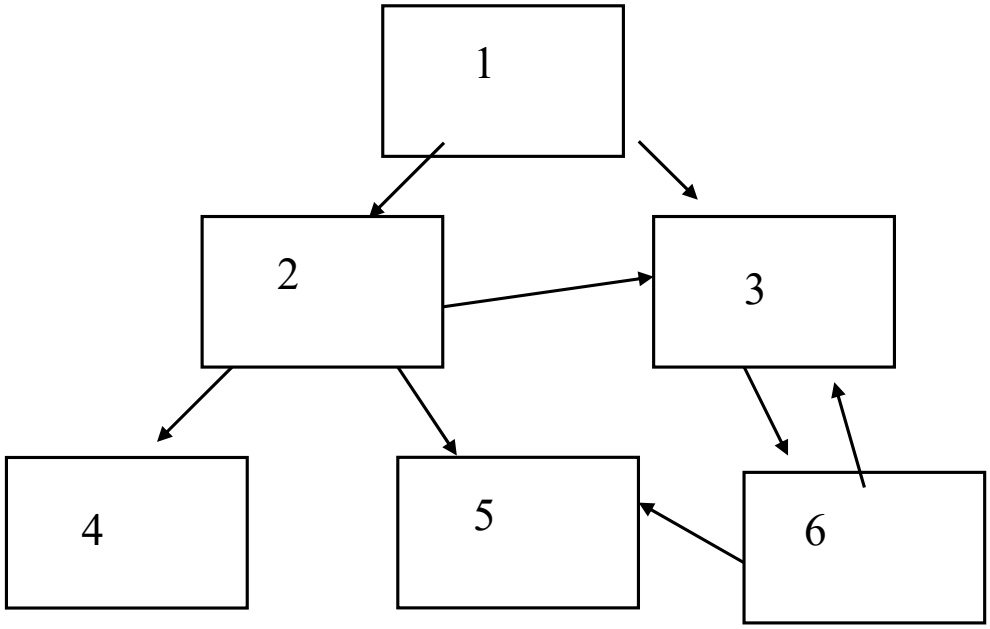


Albero binario

```
struct el  {int info;
            struct el *sinistro;
            struct el *destro;};
struct el *lista=NULL;
```

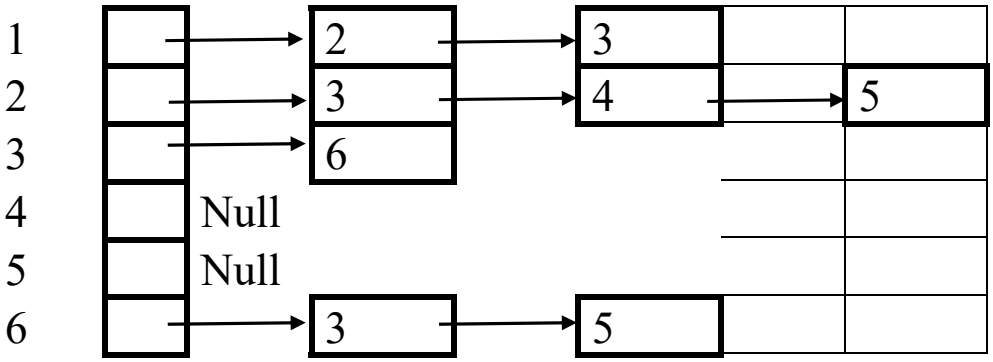


Grafo



Puntatori negli elementi – quanti?

Liste dei successori

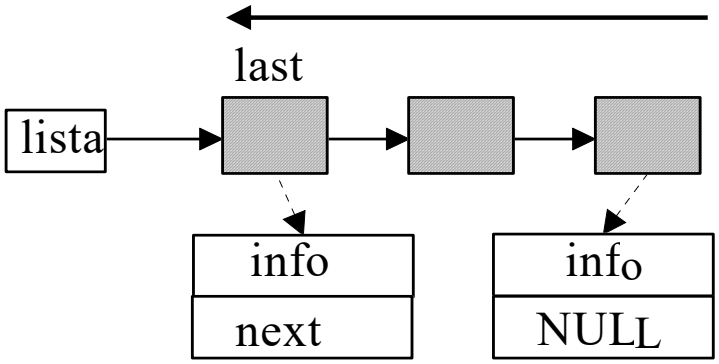


Quale scegliere in base all'applicazione?

- Gestione di una sequenza di valori (vettore)
- Gestione coda
- Gestione pila
- Indice di accesso ai file

……..

La gestione di una pila/stack con una lista monodirezionale



Strutture dati necessarie (supponiamo di averle nell'area dati globale)

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

struct el *lista=NULL;    //lista

struct el *elemento;        //singolo elemento
```

Creazione elemento nuovo, caricamento contenuto
e sua restituzione

Da decidere
- Chi fornisce componente informativa? Parametro.
- se malloc fallisce? Ritorna NULL

```
struct el *creael (int v)
 {  struct el *temp;
    temp= malloc(sizeof(struct el));
    if (temp == NULL) return NULL;
    (else non serve)

     temp->info = v;  temp->next=NULL;
     return(temp);
  }
```

Es di invocazione
```
int main(){lista=creael(6);}
```

Esecuzione main

Inserimento elemento creato in testa alla lista

Da decidere
- se elemento = NULL? Si ritorna la lista ricevuta
- stato della lista: qualsiasi

Push (lista ↓ ↑ elemento ↓)

Possibili definizioni dell'interfaccia
1) struct el *push (struct el *L, struct el *e) Consigliata

```
a.  {if (e==NULL) return L;

b.   if (L ==NULL)  return(e);

c.    e->next = L;
d.    L=e;
e.    return(L);
     }
```

Invocazione senza controllo esito creael:
1. ...
2. elemento= creael(5);
3. lista = push(lista, elemento);

Esecuzione main

2) void push (struct el **L, struct el *e)

a. { if (e==NULL) return;

b. if (*L ==NULL) { *L = e; return; }

c. e->next = *L;

d. *L=e;

}

Invocazione:

1. ...
2. elemento= creael5();
3. push(&lista, elemento);

Esecuzione main

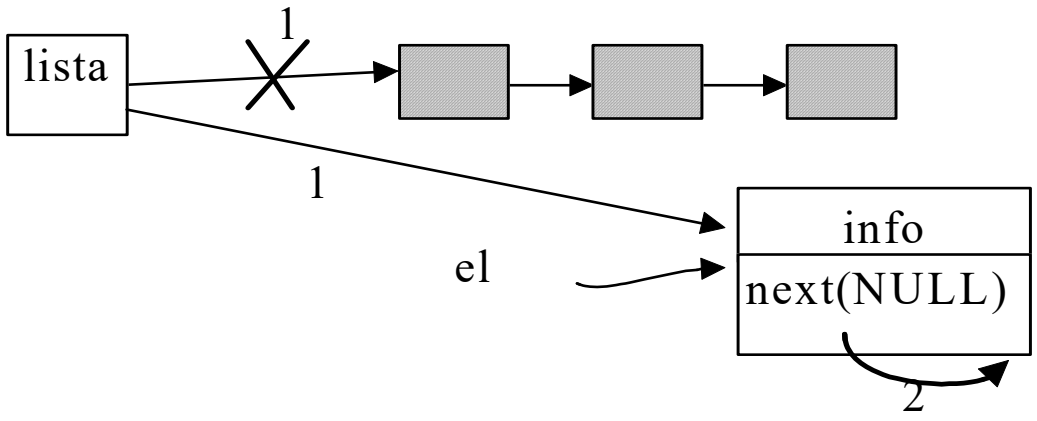
Attenzione

Passaggio parametri errato

void push (struct el *L, struct el *e)

Rispettare l'ordine delle operazioni c. e d. // Rottura lista

```
  struct el *push (struct el *e, struct el *L)
 {...
   d. L = e;
   c. e->next = L;     }
   return L;
}
```



Estrazione di un elemento dalla testa della lista

Da decidere
- stato della lista: qualsiasi – se vuota return NULL

Push (lista  elemento )

Possibili definizioni dell'interfaccia

1) struct el *pop (struct el **L)
{ struct el *temp;
a. if (*L ==NULL) return NULL;
b. temp = *L;
c. *L=temp->next;
d. return (temp);
}

Invocazione:
1. ...
2. elemento = pop (&lista);
3. if (elemento !=NULL) ...

Esecuzione main

Note:

- Si può trattare la lista vuota senza un if dedicato?
- La lista con 1 solo elemento è un caso particolare?

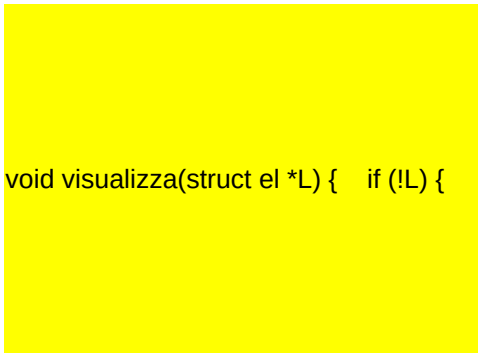
Scansione elementi di una lista monodirezionale

```
void visualizza(struct el *L)
  { while (L !=NULL)
       { printf("\ninfo: %d", L ->info);
         L = L ->next;
       }
  }
```

Invocazione: visualizza (lista);

Domande:

- Corretto con lista vuota
- Messaggio per lista vuota
- E' corretto usare L per la scansione



Un'alternativa?

```
void visualizza(struct el *L)
{ if (L!=NULL) {printf("\ninfo:%d",L->info);
                    visualizza(L->next);
                }
}
```

Codice corretto?

1.
```
void visualizza (struct el *L)
  {do
      {printf("%d ", L->info); L=L->next;}
    while (L != NULL);
  }
```

2.
```
void visualizza(struct el *L)
 { while (L->next !=NULL)
      { printf(" %d", L ->info); L = L ->next;  }
  }
```

3.
```
void visualizza (struct el *L)
  {do
      {printf("%d ", L->info); L=L->next;}
    while (L ->next != NULL);
  }
```

Ricerca nella lista (es. ricerca studente con matricola = 12)

Domanda:

- se lista è vuota -> NULL
- se elemento cercato non esiste -> NULL
- se criterio soddisfatto da più elementi -> torna il primo

Ricerca (lista ↓ valore ↓ el ↑)

Invocazione

elemento = ricerca(lista, 12);

```
struct el *ricerca (struct el *L, int valore)
   {int trovato=0;
    if (L==NULL) return NULL;

    while ((L !=NULL)&& !trovato)
      { if (L->info==valore) trovato=1;
        else L = L ->next;
       }
       if trovato return L;  else return NULL;
 }
```

Note:

- Si può trattare la lista vuota senza un if dedicato?

- Versione accettabile?

```
struct el *ricerca (struct el *L, int valore)
  {     while (L !=NULL)
           { if (L->info==valore) return L;
             L = L ->next;
           }
      return NULL;
  }
```

- Versione accettabile?

```
struct el  *Ricerca (struct el *L, int valore)
   { while (L ->info != valore)L = L–>next;
     return L;
   }
```

- Versione accettabile?

```
struct el  *Ricerca (struct el *L, int valore)
      {  while ((L ->info != valore) && (L!=NULL))  L = L–>next;
         return L;
       }
```

Ricerca ultimo che soddisfa una condizione

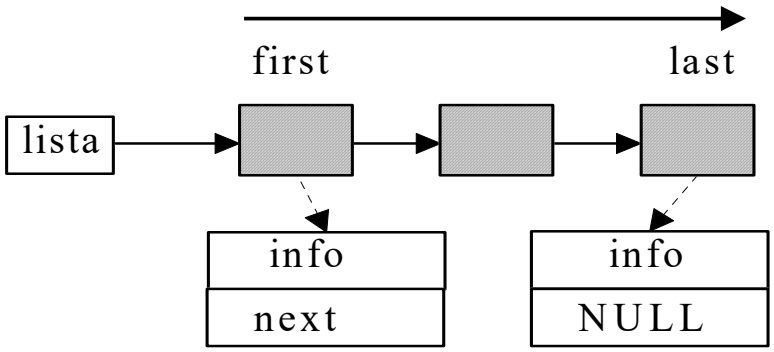
Ricerca (lista ↓ valore ↓ el ↑) Invocazione

elemento = ricerca(lista, 12);

```
struct el *ricerca (struct el *L, int valore)
   {struct el *temp=NULL;

     while (L !=NULL)
       { if (L->info==valore) temp=L;
         L = L ->next;
        }
       return temp;
 }
```

- Se volessi estrarre tutti quelli che soddisfano?

La gestione di una coda con una lista monodirezionale



Inserimento elemento in coda alla lista

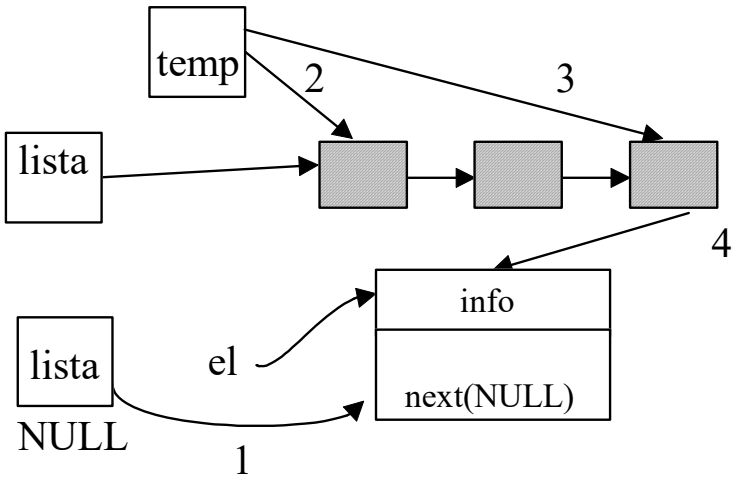
Insert (lista ↓ ↑(?) , el ↓) Invocazione

lista=insert(elemento,lista)

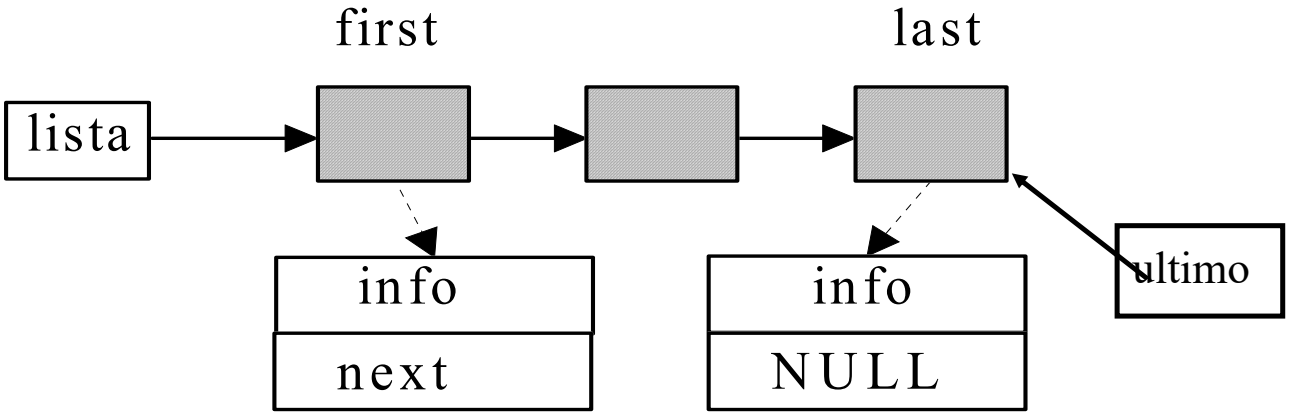
Domande

- Stato della lista qualsiasi
- Se elemento =NULL ritorna lista ricevuta
- Ins in coda richiede di ritornare la lista cambiata
- temp è necessaria?

```
struct el *insertincoda struct el *e, struct el *L)
{ struct el *temp;
  if (e==NULL) return L
  if (L == NULL)  return(e);

  temp = L;     //temp deve puntare a ultimo
  while (temp->next !=NULL) temp=temp->next;

  temp->next=e;

   return(L)
 }
```



Come migliorare gli inserimenti in coda?



Lista vuota?

Dopo inserimento primo elemento?

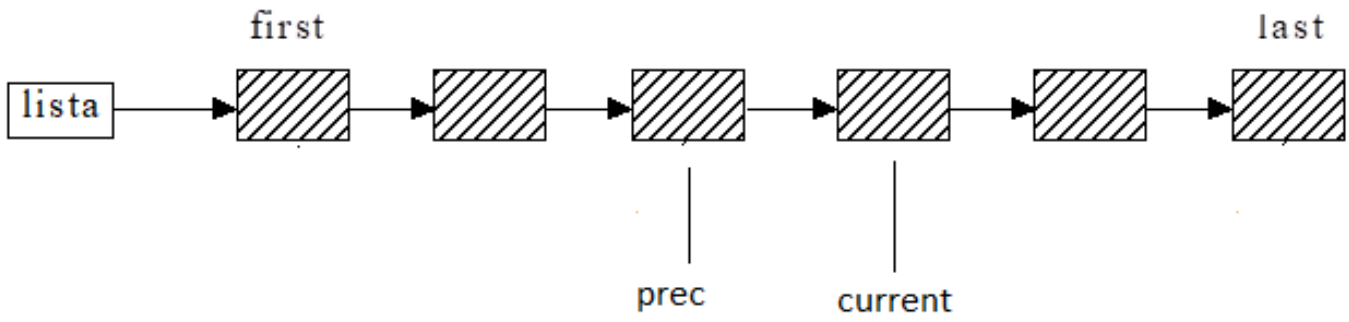
Inserimento dal secondo in poi?

Estrarre dalla lista primo elemento con valore 5 e restituire lista modificata e l'indirizzo dell'elemento

⇓

Domande
a. Lista vuota? -> NULL
b. Elemento non esiste -> NULL
oppure
Elemento da estrarre può trovarsi in:
c. posizione intermedia
d. prima posizione
e. ultima posizione

Casi nuovi?
f. Elemento selezionato unico nella lista



struct el *estrai(struct el *L, int num, struct el **e) ;

Invocazione

```
#include <stdio.h>
struct el {int info; struct el *next; };    struct el *lista=NULL, *elemento;
int numero;
int main() {… lista= estrai(lista, 5, &elemento);... }
```

Implementazione

```
struct el *estrai(struct el *L, int num, struct el **e)
{     struct el *current=L, *prec=NULL; // per scansione

//caso a.
      if (current==NULL) {*e=NULL; return NULL;}  //lista vuota

      while (current !=NULL)
        { if (current->info==num)   //lo trova
              { if (current==L) // first caso d.
                    {   L=current->next;*e=current; return L;}
                else //intermedio caso c. ?
                  {prec->next=current->next; *e=current; return L;}
             }
          else //non lo trova e passa al prossimo
              {prec=current; current= current->next; }
        }   //end while

       *e=NULL;          return L;  //non trovato   caso b.
}
```

Funzione “cancella” estrae dalla lista e cancella dalla memoria tutti gli elementi della lista con valore 5

Data la funzione
struct el *estrai(struct el *L, int num, struct el **e)
/*Estrae nel parametro **e indirizzo primo elemento della lista L con valore num (se non esiste Null) e restituire come valore ritornato la lista modificata */

Prototipo
struct el *cancella (struct el *Lis, int Val)

Invocazione
int main (){lista = cancella (lista, 5); }

Implementazione

```
struct el *cancella (struct el *Lis, int Val)
{ struct el *elem;
    Lis=estrai(Lis, Val, &elem);
    While (elem!=NULL)
        {free(elem);
         Lis=estrai(Lis, Val, &elem);
        }

}
```

Funzione INS che riceve un valore integer, crea un elemento dinamico nel quale memorizza il valore. Poi inserisce l'elemento dopo il primo della lista che contiene il valore 5.

Domande

- Lista vuota
- Lista senza elemento con valore 5

struct el * InsDopoVal (struct el *L, int valnew, int valel)

/*riceve la lista L che restituice modificata come valore di ritorno valnew che viene inserito nel novo elemento creato. Ricerca poi la posizione dopo il primo che ha il vlore valel
*/

Funzioni utili:
struct el *creael(int valnew);
seguita da verifica effettiva creazione dell'elemento

struct el * RicercaPrimovalel(struct el *L, int valel);
/* restituisce indirizzo del primo elemento della lista L con valore valel e ne restituisce l'indirizzo come valore ritornato (NULL se non esiste)
Seguita da verifica di averlo trovato*/