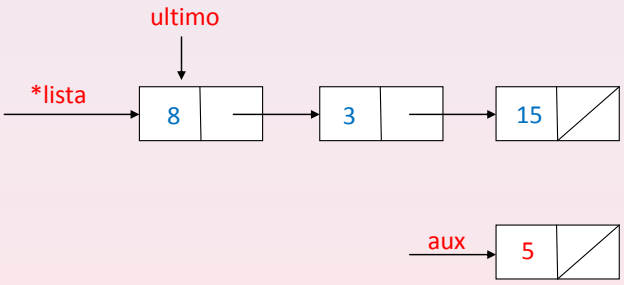
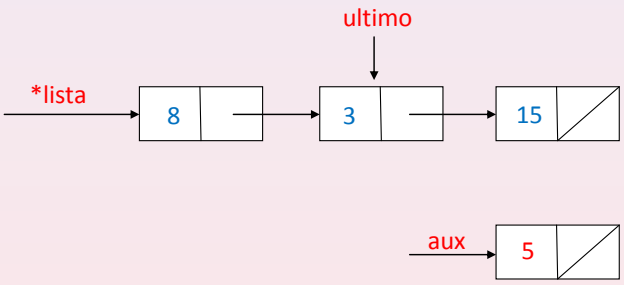
Inserimento di un elemento in coda

- Se la lista è vuota coincide con l'inserimento in testa
  $\Longrightarrow$ è necessario il passaggio per indirizzo!
- Se la lista non è vuota, bisogna scandirla fino in fondo
  $\Longrightarrow$ dobbiamo usare un puntatore ausiliario per la scansione
- La scansione deve terminare in corrispondenza dell'ultimo elemento al quale va collegato quello nuovo



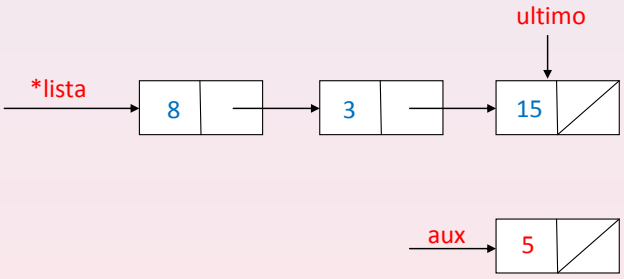
Inserimento di un elemento in coda

- Se la lista è vuota coincide con l'inserimento in testa
  $\Longrightarrow$ è necessario il passaggio per indirizzo!
- Se la lista non è vuota, bisogna scandirla fino in fondo
  $\Longrightarrow$ dobbiamo usare un puntatore ausiliario per la scansione
- La scansione deve terminare in corrispondenza dell'ultimo elemento al quale va collegato quello nuovo



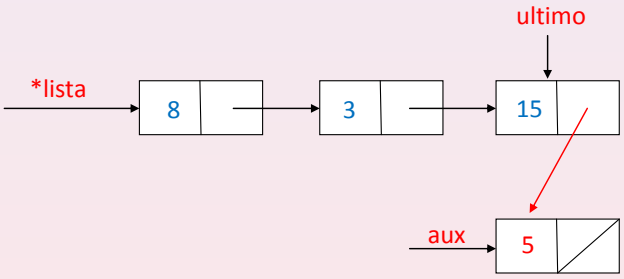
Inserimento di un elemento in coda

- Se la lista è vuota coincide con l'inserimento in testa
  $\Longrightarrow$ è necessario il passaggio per indirizzo!
- Se la lista non è vuota, bisogna scandirla fino in fondo
  $\Longrightarrow$ dobbiamo usare un puntatore ausiliario per la scansione
- La scansione deve terminare in corrispondenza dell'ultimo elemento al quale va collegato quello nuovo



Inserimento di un elemento in coda

- Se la lista è vuota coincide con l'inserimento in testa
  $\Longrightarrow$ è necessario il passaggio per indirizzo!
- Se la lista non è vuota, bisogna scandirla fino in fondo
  $\Longrightarrow$ dobbiamo usare un puntatore ausiliario per la scansione
- La scansione deve terminare in corrispondenza dell'ultimo elemento al quale va collegato quello nuovo



Codice della versione iterativa

```
void InserzioneInCoda(ListaDiElementi *lista, TipoElementoLista elem)
{
  ListaDiElementi ultimo;   /* puntatore usato per la scansione */
  ListaDiElementi aux;
                            /* creazione del nuovo elemento */
  aux = malloc(sizeof(ElementoLista));
  aux->info = elem;
  aux->next = NULL;

  if (*lista == NULL)
    *lista = aux;
  else {
    ultimo = *lista;
    while (ultimo->next != NULL)
      ultimo = ultimo->next;
                     /* concatenazione del nuovo elemento in coda alla lista */
    ultimo->next = aux;
  }
}
```


Inserimento ricorsivo di un elemento in coda

- Caratterizzazione induttiva dell'inserimento in coda

  Sia nuovaLista la lista ottenuta inserendo elem in coda a lista.

  1. se lista è vuota, allora nuovaLista è costituita dal solo `elem` (caso base)
  2. altrimenti nuovaLista è ottenuta da lista facendo l'inserimento di elem in coda al resto di lista (caso ricorsivo)

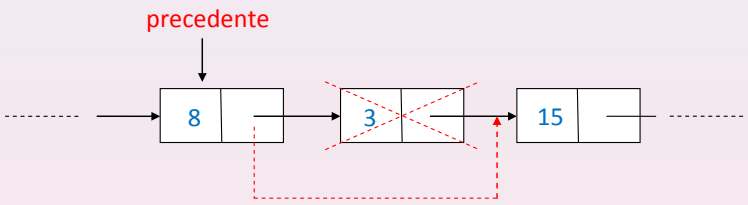
- Esercizio: Scrivere il codice corrispondente.

Cancellazione della prima occorrenza di un elemento

- si scandisce la lista alla ricerca dell'elemento
- se l'elemento non compare non si fa nulla
- altrimenti, a seconda di dove si trova l'elemento, si distinguono tre casi
  1. l'elemento è il primo della lista: si aggiorna il puntatore iniziale in modo che punti all'elemento successivo
     $\Longrightarrow$ passaggio per indirizzo!!
  2. l'elemento non è né il primo né l'ultimo: si aggiorna il campo next dell'elemento che precede quello da cancellare in modo che punti all'elemento che segue
  3. l'elemento è l'ultimo: come (2), solo che il campo next dell'elemento precedente viene posto a `NULL`
- in tutti e tre i casi bisogna liberare la memoria occupata dall'elemento da cancellare

Osservazioni:

- per poter aggiornare il campo next dell'elemento precedente, bisogna fermare la scansione sull'elemento precedente (e non su quello da cancellare)



- per fermare la scansione dopo aver trovato e cancellato l'elemento, si utilizza una sentinella booleana

```
void CancellaElementoLista(ListaDiElementi *lista, TipoElementoLista elem)
{
  ListaDiElementi prec;    /* puntatore all'elemento precedente */
  ListaDiElementi corr;    /* puntatore all'elemento corrente */
  boolean trovato;        /* usato per terminare la scansione */

  if (*lista != NULL)
    if ((*lista)->info==elem)
        { /* cancella il primo elemento */
            CancellaPrimo(lista);
        }
    else    /* scansione della lista e cancellazione dell'elemento */
            prec = *lista; corr = prec->next; trovato = false;
            while (corr != NULL && !trovato)
              if (corr->info == elem)
                {   /* cancella l'elemento */
                    trovato = true;         /* provoca l'uscita dal ciclo */
                    prec->next = corr->next;
                    free(corr);  }
              else  {
                    prec = prec->next;    /* avanzamento dei due puntatori */
                    corr = corr->next;   }
}
```


Versione ricorsiva:

```
void CancellaElementoLista(ListaDiElementi *lista, TipoElementoLista elem)
{
  if (*lista != NULL)
    if ((*lista)->info== elem)
        { /* cancella il primo elemento */
            CancellaPrimo(lista);
        }
    else   /* cancella elem dal resto */
        CancellaElementoLista(&((*lista)->next), elem);
}
```

Cancellazione di tutte le occorrenze di un elemento

Versione iterativa

- analoga alla cancellazione della prima occorrenza
- però, dopo aver trovato e cancellato l'elemento, bisogna continuare la scansione
- ci si ferma solo quando si è arrivati alla fine della lista

  $\Longrightarrow$ non serve la sentinella booleana per fermare la scansione

Cancellazione di tutte le occorrenze di un elemento

Caratterizzazione induttiva

Sia ris la lista ottenuta cancellando tutte le occorrenze di elem da lista. Allora:

1. se lista è la lista vuota, allora ris è la lista vuota (caso base)
2. altrimenti, se il primo elemento di lista è uguale ad elem, allora ris è ottenuta da lista cancellando il primo elemento e tutte le occorrenze di elem dal resto di lista (caso ricorsivo)
3. altrimenti ris è ottenuta da lista cancellando tutte le occorrenze di elem dal resto di lista (caso ricorsivo)

Esercizio

Implementare le due versioni

Versione iterativa

```
void CancellaTuttiLista(ListaDiElementi *lista, TipoElementoLista elem)
{
  ListaDiElementi prec;    /* puntatore all'elemento precedente */
  ListaDiElementi corr;    /* puntatore all'elemento corrente */
  boolean trovato = false;
  while ((*lista != NULL) && !trovato)  /* cancella le occorrenze */
    if ((*lista)->info!=elem)           /* di elem in testa       */
        trovato = true;
    else  CancellaPrimo(lista);

   if (*lista != NULL)
        {
            prec = *lista; corr = prec->next;
            while (corr != NULL)
              if (corr->info == elem)
                {   /* cancella l'elemento */
                    prec->next = corr->next;
                    free(corr);
                    corr = prec->next;}
              else  {
                    prec = prec->next;    /* avanzamento dei due puntatori */
                    corr = corr->next;   }
        }
}
```


Versione ricorsiva

```
void CancellaTuttiLista(ListaDiElementi *lista, TipoElementoLista elem)
{
  ListaDiElementi aux;

  if (*lista != NULL)
    if ((*lista)->info==elem)
    {                                /* cancellazione del primo elemento */
      CancellaPrimo(lista);
                              /* cancellazione di elem dal resto della lista */
      CancellaTuttiLista(lista, elem);
    }
    else
      CancellaTuttiLista(&((*lista)->next), elem);
}
```

Inserimento di un elemento in una lista ordinata

- Data una lista (ad es. di interi) già ordinata (in ordine crescente), si vuole inserire un nuovo elemento mantenendo l'ordinamento.

  Versione iterativa: per **esercizio**

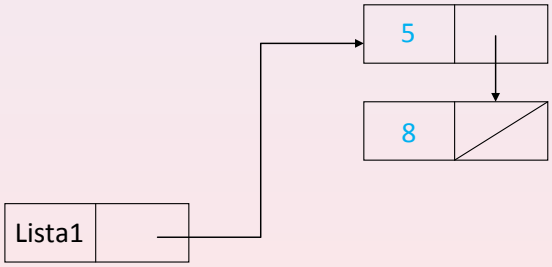
  Versione ricorsiva

- Caratterizziamo il problema induttivamente
- Sia ris la lista ottenuta inserendo l'elemento elem nella lista ordinata lista.
  1. se lista è la lista vuota, allora ris è costituita solo da elem (caso base)
  2. se il primo elemento di lista è maggiore o uguale a elem, allora ris è ottenuta da lista inserendo elem in testa (caso base)
  3. altrimenti ris è ottenuta da lista inserendo ordinatamente elem nel resto di lista (caso ricorsivo)


```
void InserzioneOrdinata(ListaDiElementi *lista, int elem)
{
if (*lista == NULL)
   InserisciTestaLista(lista, elem);
else
    if ((*lista) --> info >= elem)
        InserisciTestaLista(lista, elem);
    else
        InserzioneOrdinata(&((*lista)->next), elem);
}
```

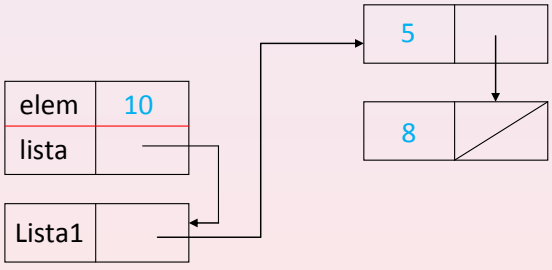
InserzioneOrdinata(&Lista1, 10)

PILA HEAP



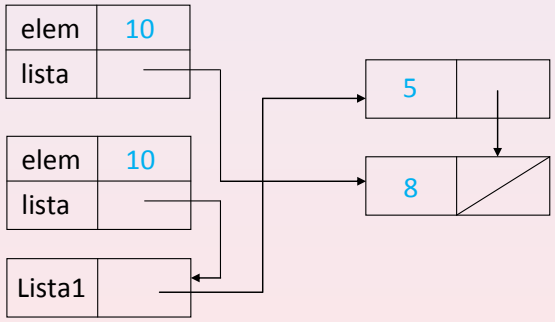
InserzioneOrdinata(&Lista1, 10)

PILA HEAP



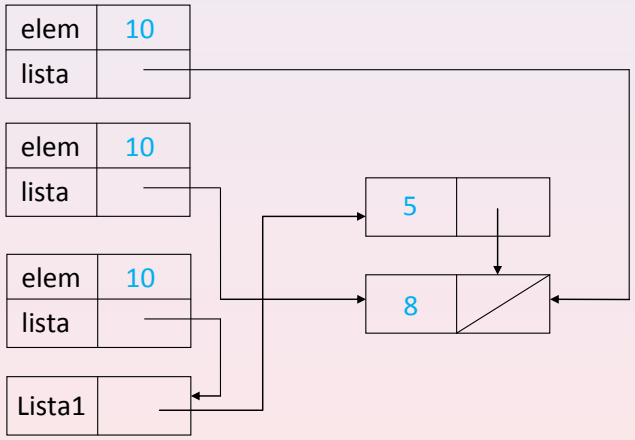
InserzioneOrdinata(&Lista1, 10)

PILA HEAP

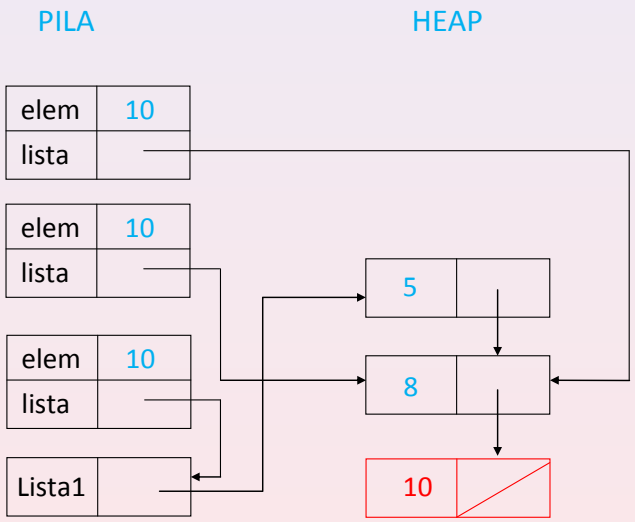


InserzioneOrdinata(&Lista1, 10)

PILA HEAP



InserzioneOrdinata(&Lista1, 10)




```
void InserzioneOrdinata(ListaDiElementi *lista, int elem)
{
if (*lista == NULL)
   InserisciTestaLista(lista, elem);
else
    if ((*lista) --> info >= elem)
        InserisciTestaLista(lista, elem);
    else
        {
          prec = *lista;
          corr = prec -> next;
          trovato = false;
          while (corr!=NULL && !trovato)
             {
                if (corr -> info >= elem)
                   trovato = true;
                else
                   {prec = prec -> next;
                    corr = corr -> next;}
             }
           aux = malloc(sizeof(ElementoLista));
           aux -> info = elem;
           prec -> next = aux;
           aux -> next = corr;
}
```