Sequenze lineari

- Insieme finito di elementi disposti consecutivamente
- Ogni elemento ha associato un indice di posizione (univoco)
  - L'ordine è importante
- L'operazione più elementare è l'accesso ai singoli elementi

Modalità di Accesso

- Accesso diretto (array)
  - Accediamo direttamente all'elemento a_i senza dover attraversare la sequenza.
- Accesso sequenziale (liste)
  - Raggiungiamo l'elemento attraversando la sequenza a partire da un suo estremo.
  - Costo O(i)
  - Accedere a a_{i+1} (da a_i) costa O(1)

Allocazione della Memoria

- Array e liste corrispondono a modi diversi di allocare la memoria
- Array: le locazioni di memoria corrispondenti ad elementi contigui sono contigue.
  - Il nome dell'array corrisponde alla locazione del primo elemento (a[0])
  - Se x è l'indirizzo di a, a[0] si trova all'indirizzo x
  - a[1] si trova all'indirizzo x+4, ecc.

Allocazione della Memoria

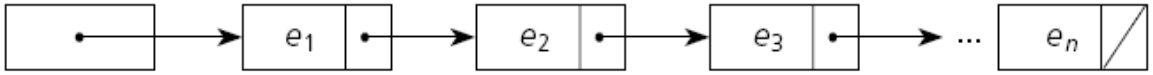
- Liste: gli elementi si trovano in locazioni di memoria non necessariamente contigue.
  - Ogni elemento memorizza, oltre al proprio valore, l'indirizzo del valore successivo.
- **Vantaggio**: le liste si prestano bene a implementare sequenze dinamiche.

Array di dimensione variabile

- E’ necessario poter ridimensionare l’array (allocarne uno nuovo di diversa dimensione)
- L’operazione richiede O(n) tempo per ciascuna nuova allocazione
- Si può fare meglio?
  - Costo cumulativo O(n) per ciascun gruppo di Ω(n) operazioni consecutive
  - Costo (distribuito): O(1) per operazione

liste

- una **lista concatenata** è una sequenza di elementi collegati l'uno all'altro da un puntatore
- gli elementi si compongono cioè di due parti: una che contiene l'informazione e l'altra costituita da un puntatore al successivo elemento



- *Liste semplici:* ogni elemento contiene un unico puntatore che lo collega al nodo successivo
- *Liste doppiamente concatenate:* ogni elemento contiene due puntatori, uno all'elemento precedente e l'altro al successivo
- *Liste circolari semplici:* l'ultimo elemento si collega al primo in modo che la lista può essere attraversata in modo circolare.
- *Liste circolari doppiamente concatenate:* l'ultimo elemento si collega al primo elemento e viceversa

dichiarazione di un elemento

- si può definire un elemento, o "nodo" della lista, mediante i costrutti `struct` o `class`

```
struct nodo {
 int dato;
 nodo* puntatore;
};
```

```
class nodo {
public :
 int dato;
 nodo* puntatore;
};
```


costruzione di una lista

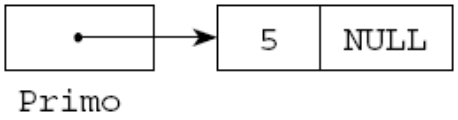
- la creazione di una lista concatenata implica i seguenti passi:

Passo 1. Dichiarare il tipo del nodo ed il puntatore di testa
Passo 2. Allocare memoria per un nodo utilizzando l'operatore `new` assegnandone l'indirizzo ad un puntatore
Passo 3. Creare iterativamente il primo elemento ed i successivi
Passo 4. Ripetere finchè vi siano nodi da immettere

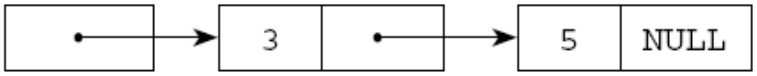
```
class Elemento {
public :
   Elemento* suc;
   int dato;
   Elemento (Elemento* n, int d): suc(n), dato(d) {}
};
```


costruzione di una lista

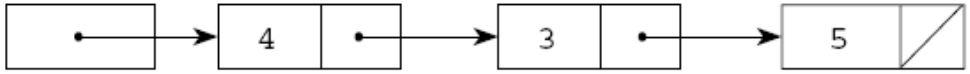

```
Elemento* Primo = NULL; // lista vuota
Primo = new Elemento(Primo, 5); // primo elemento
```




```
Primo = new Elemento(Primo, 3);
```



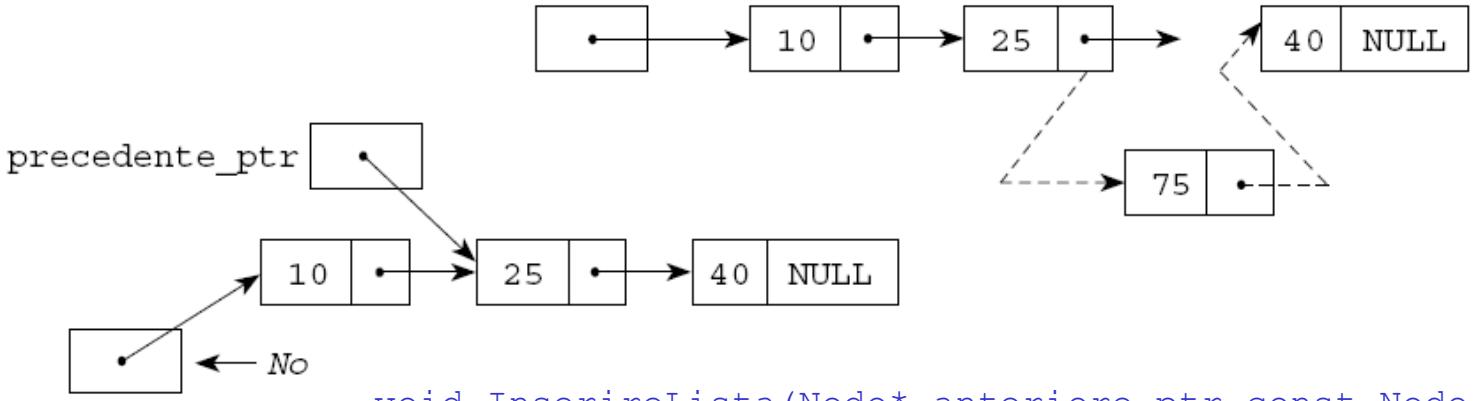

```
Primo = new Elemento(Primo, 4);
```

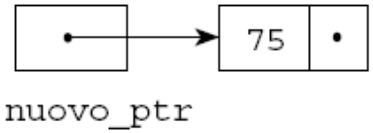


inserimento in testa

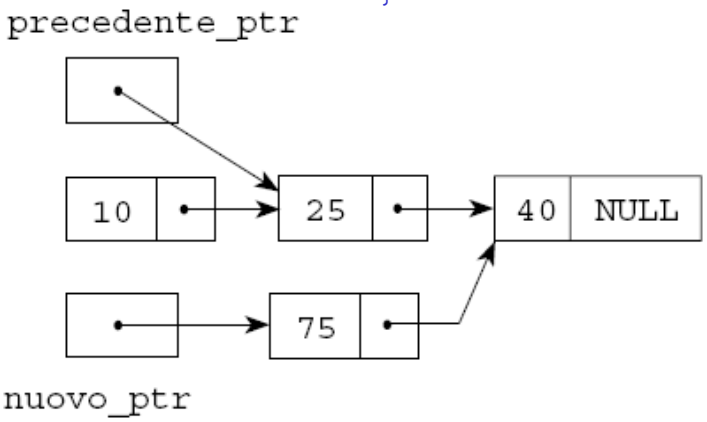

```
void InserTestaLista (Nodo& testa_ptr, const Nodo::Item& inform)
{
   Nodo * nuovo_ptr;
   nuovo_ptr = new Nodo;         // assegna nuovo nodo
   nuovo_ptr->dato = inform;     // mette elemento in nuovo nodo
   nuovo_ptr->suc = testa_ptr ; // collega nuovo nodo
                                  // al fronte della lista
   testa_ptr = nuovo_ptr ;  // muove puntatore testa
                            // e punta al nuovo nodo
}
```

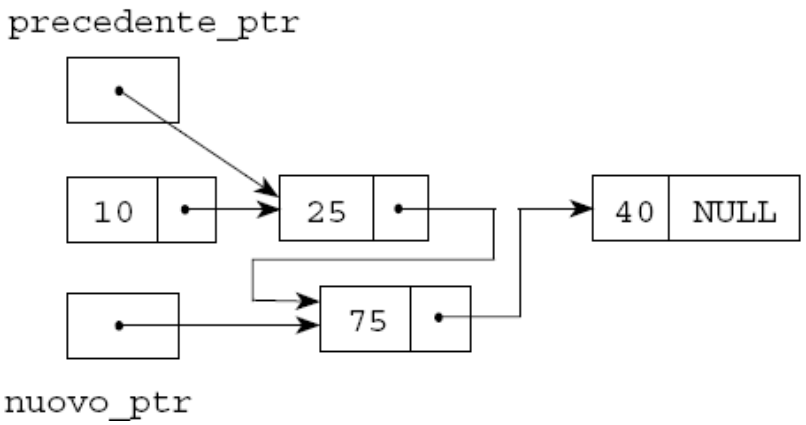

inserimento in un posto predeterminato






```
void InserireLista(Nodo* anteriore_ptr,const Nodo::Item& inform)
{
   Nodo* nuovo_ptr;
   nuovo_ptr = new Nodo;
   nuovo_ptr->info = inform;
   nuovo_ptr->suc = ant_ptr->suc;
   ant_ptr->suc = nuovo_ptr;
}
```





ricerca di un elemento

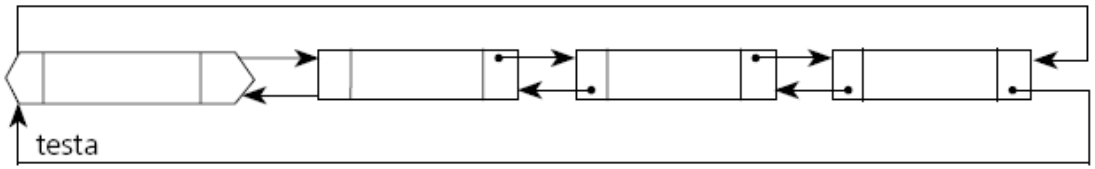
- Molto simile alla ricerca in array
- Semplicemente alla fine viene restituito un puntatore all'elemento trovato

rimozione di un elemento

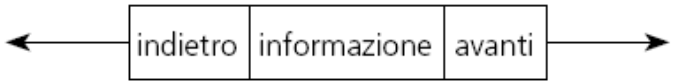
1. cercare il nodo che contiene il dato, farlo puntare da `pos` e far puntare da `ant` il nodo che lo precede (se non è il primo)
2. mettere il puntatore `suc` del nodo puntato da `ant` al puntatore `suc` del nodo puntato da `pos`
3. se `pos` è il puntatore di testa si mette `p` al campo `suc` del nodo puntato da `pos`
4. si libera la memoria occupata dal nodo puntato da `pos`

```
void ListaS::Cancellare(Telemento x)
{
    Nodo* ant=NULL, *pos =p;
      // ricerca del nodo da cancellare
    ......
      // si cancellerà il nodo puntato da pos
    if(ant!=NULL)
       ant->Ssuc(pos->Osuc());    // non è il primo
    else
        p = pos->Osuc(); // è il primo
    pos->Ssuc(NULL);
    delete pos;
 }
```


lista doppiamente concatenata



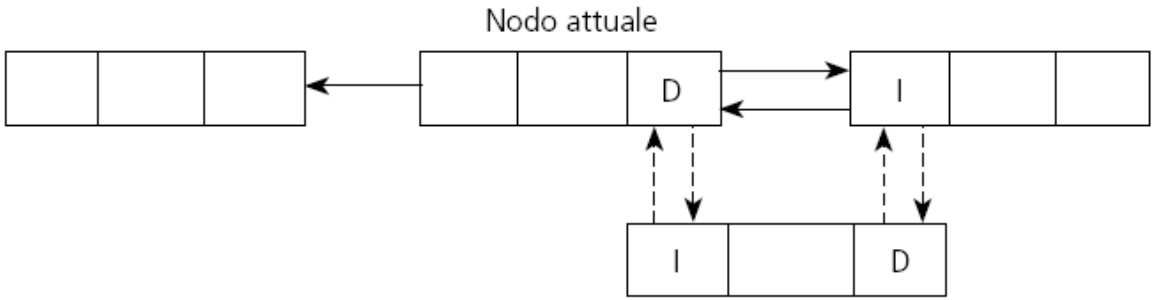
(*a*)



(*b*)

```
class Elemento
{
   public :
   Elemento *avanti, *indietro;
   int dato;                          // dato di tipo intero
   Elemento (Elemento *f = 0, Elemento *b = 0, int d = 0)
     : avanti(f), indietro(b), dato(d) {}
};
```


inserimento in lista doppiamente concatenata



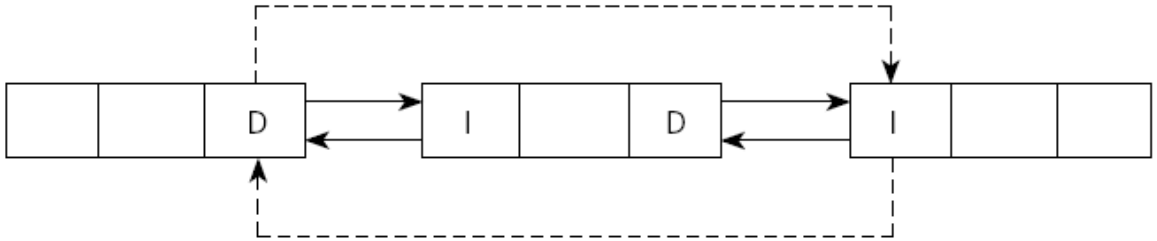
inserimento in testa a lista doppiamente concatenata

1. allocare dinamicamente un nuovo nodo, assegnarne l'indirizzo ad un puntatore nuovo e copiarvi dentro l'elemento e che si vuole inserire
2. assegnare il campo suc del nuovo nodo al puntatore di testa p, ed il campo ant del primo nodo, se esiste, al nodo nuovo. Se la lista è vuota non fare alcunché
3. far puntare p al nodo nuovo

inserimento in una determinata posizione in lista doppiamente concatenata

1. cercare la posizione dove bisogna inserire il nodo
2. allocare dinamicamente memoria al nuovo nodo puntato dal puntatore nuovo, e copiarvi il contenuto informativo `e`
3. far puntare il campo `suc` del nuovo nodo al nodo `pos` che va dopo la posizione del nuovo nodo (oppure a `NULL` in caso non vi sia alcun nodo dopo la nuova posizione); l'attributo `ant` del nodo successivo a quello che occupa la posizione del nuovo nodo che è `pos`, deve puntare a `nuovo` se esiste; se non esiste non fare alcunché
4. far puntare l'attributo `suc` del puntatore `ant` al nuovo nodo; l'attributo `ant` del nuovo nodo, farlo puntare ad `ant`

rimozione da lista doppiamente concatenata



1. ricerca del nodo che contiene il dato; si deve avere l'indirizzo del nodo da rimuovere e l'indirizzo dell'antecedente (`ant`).
2. l'attributo `suc` del nodo anteriore (`ant`) deve puntare all'attributo `suc` del nodo da rimuovere, `pos` (se non è il primo nodo della lista); se è il primo della lista l'attributo `p` deve puntare all'attributo `suc` del nodo da rimuovere `pos`
3. l'attributo `ant` del nodo successivo a quello da cancellare deve puntare all'attributo `ant` del nodo da rimuovere (se non è l'ultimo nodo della lista); se è l'ultimo nodo della lista non fare alcunché
4. si libera la memoria occupata dal nodo rimosso `pos`

inserimento in lista circolare

1. allocare memoria al nodo `nuovo` e riempirlo di informazione
2. se la lista è vuota far puntare il `succ` di `nuovo` al nodo stesso, e far puntare il puntatore di lista al `nuovo`
3. se la lista non è vuota si deve decidere dove collocare il `nuovo`, conservando l'indirizzo del nodo antecedente `ant`; collegare l'attributo `suc` di `nuovo` con l'attributo `suc` del nodo antecedente `ant`; collegare l'attributo `suc` del nodo antecedente `ant` con il `nuovo`; se si pretende che il `nuovo` già inserito sia il primo della lista circolare, muovere il puntatore della lista circolare al `nuovo`, altrimenti non fare alcunché

rimozione da lista circolare

1. cercare il nodo `ptrnodo` che contiene il dato conservando un puntatore all'antecedente `ant`
2. far puntare il campo `suc` del nodo antecedente `ant` dove punta il campo `suc` del nodo da cancellare; se la lista conteneva un solo nodo si mette a `NULL` il puntatore `p` della lista
3. se il nodo da rimuovere è quello puntato dal puntatore d'accesso alla lista, `p`, e la lista contiene più di un nodo, si modifica `p` per mandarlo dove punta il campo `suc` del nodo puntato da `p` (se la lista rimane vuota fare prendere a `p` il valore `NULL`).
4. da ultimo, si libera la memoria occupata dal nodo eliminato

