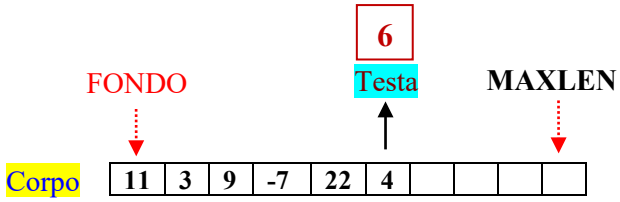
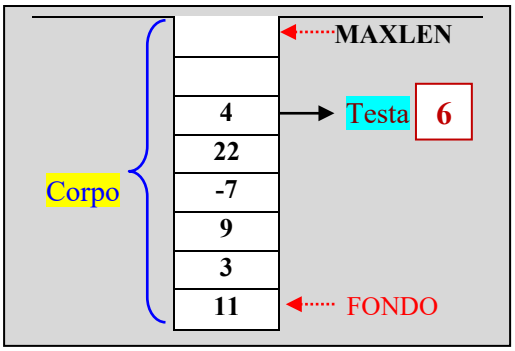
STRUTTURA DATI ASTRATTA LINEARE PILA

IMPLEMENTATA CON UNA STRUTTURA DATI SEQUENZIALE, AD ALLOCAZIONE STATICA ED AD ACCESSO DIRETTO (OSSIA CON UN VETTORE O ARRAY)

```
TIPO PILA = RECORD
              Corpo : ARRAY[MAXLEN] DI INT
              Testa : INT
            FINE RECORD
```



P_1 = [11, 3, 9, -7, 22, 4)




```
/* funzione ADT Crea ( ) */
FUNZIONE Crea ( ) : PILA
p: PILA
i: INT
INIZIO
PER i ← 1 A MAXLEN ESEGUI
  p.Corpo[i] ← 0
  i ← i + 1
FINE PER
p.Testa ← 0
RITORNA (p)
FINE
```

```
/* funzione ADT TestVuota ( ) */
FUNZIONE TestVuota (VAL p: PILA) : BOOL
esito: BOOL
INIZIO
SE (p.Testa = 0)
  ALLORA
    esito ← VERO
  ALTRIMENTI
    esito ← FALSO
FINE SE
RITORNA (esito)
FINE
```

```
/* funzione di servizio StampaPila ( ) */
PROCEDURA StampaPila (VAL p:PILA)
i: INT
INIZIO
SE (TestVuota (p) = FALSO)
  ALLORA
    PER i ← 1 A p.Testa ESEGUI
      Scrivi(p.Corpo[i])
      i ← i + 1
    FINE PER
    Scrivi(p.Testa)
  ALTRIMENTI
    Scrivi ("PILA vuota!")
FINE SE
RITORNA
FINE
```

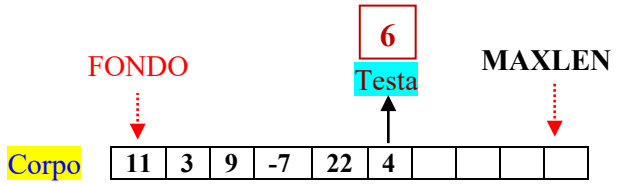
```
/* funzione ADT Pop ( ) */
FUNZIONE Pop (REF ele: INT, REF p:PILA) : BOOL
esito: BOOL
INIZIO
SE (TestVuota (p) = FALSO)
  ALLORA
    ele ← p.Corpo[p.Testa]
    p.Testa ← p.Testa – 1
    esito ← VERO
  ALTRIMENTI
    Scrivi ("PILA vuota!")
    esito ← FALSO
FINE SE
RITORNA (esito)
FINE
```

```
/* funzione ADT Push ( ) */
PROCEDURA Push (VAL ele: INT, REF p:PILA)
INIZIO
SE (p.Testa < MAXLEN)
  ALLORA
    p.Testa ← pTesta + 1
    p.Corpo[p.Testa] ← ele
  ALTRIMENTI
    Scrivi ("PILA piena!")
FINE SE
RITORNA
FINE
```

STRUTTURA DATI ASTRATTA LINEARE CODA

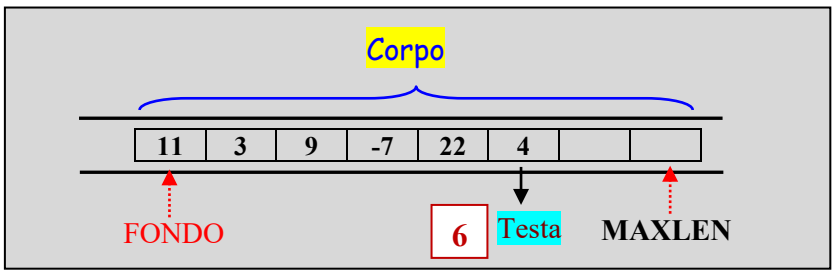
IMPLEMENTATA CON UNA STRUTTURA DATI SEQUENZIALE, AD ALLOCAZIONE STATICA ED AD ACCESSO DIRETTO (OSSIA CON UN VETTORE O ARRAY)

```
TIPO CODA = RECORD
              Corpo : ARRAY[MAXLEN] DI INT
              Testa : INT
            FINE RECORD
```



$C_1 = \{11, 3, 9, -7, 22, 4)$

```
/* funzione ADT Crea ( ) */
FUNZIONE Crea ( ) : CODA
c: CODA
i: INT
INIZIO
PER i ← 1 A MAXLEN ESEGUI
  c.Corpo[i] ← 0
  i ← i + 1
FINE PER
c.Testa ← 0
RITORNA (c)
FINE
```




```
/* funzione ADT TestVuota ( ) */
FUNZIONE TestVuota (VAL c: CODA) : BOOL
esito: BOOL
INIZIO
SE (c.Testa = 0)
  ALLORA
    esito ← VERO
  ALTRIMENTI
    esito ← FALSO
FINE SE
RITORNA (esito)
FINE
```

```
/* funzione di servizio StampaCoda ( ) */
PROCEDURA StampaCoda (VAL c:CODA)
i: INT
INIZIO
SE (TestVuota (c) = FALSO)
  ALLORA
    PER i ← 1 A c.Testa ESEGUI
      Scrivi(c.Corpo[i])
      i ← i + 1
    FINE PER
    Scrivi(c.Testa)
  ALTRIMENTI
    Scrivi ("CODA vuota!")
FINE SE
RITORNA
FINE
```

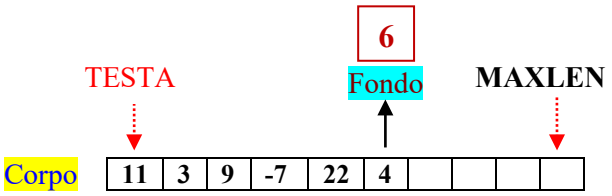
```
/* funzione ADT Inserisci ( ) */
PROCEDURA Inserisci (VAL ele: INT, REF c:CODA)
INIZIO
SE (c.Testa < MAXLEN)
  ALLORA                          //shift totale a dx
    PER i ← (c.Testa + 1) INDIETRO A 2 ESEGUI
      c.Corpo[i] ← c.Corpo[i - 1]
      i ← i – 1
    FINE PER
    c.Corpo[1] ← ele              //insert elemento in testa
    c.Testa ← c.Testa + 1
  ALTRIMENTI
     Scrivi ("Coda piena!")
FINE SE
RITORNA
FINE
```

```
/* funzione ADT Estrai ( ) */
FUNZIONE Estrai (REF ele: INT, REF c:CODA) : BOOL
esito: BOOL
i : INT
INIZIO
SE (TestVuota (c) = FALSO)
  ALLORA
    ele ← c.Corpo[c.Testa]
    c.Testa ← c.Testa - 1
    esito ← VERO
  ALTRIMENTI
    Scrivi ("CODA vuota!")
    esito ← FALSO
FINE SE
RITORNA (esito)
FINE
```

STRUTTURA DATI ASTRATTA LINEARE LISTA o SEQUENZA

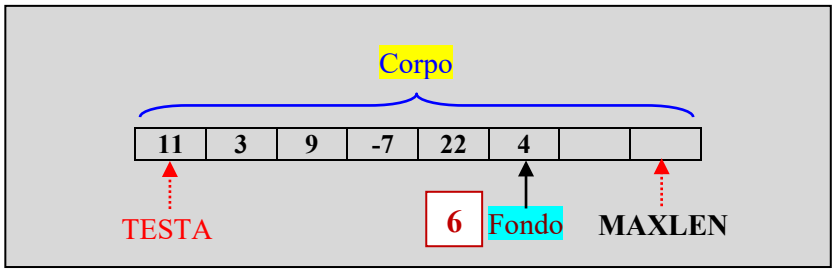
IMPLEMENTATA CON UNA STRUTTURA DATI SEQUENZIALE, AD ALLOCAZIONE STATICA ED AD ACCESSO DIRETTO (OSSIA CON UN VETTORE O ARRAY)

```
TIPO LISTA = RECORD
               Corpo : ARRAY[MAXLEN] DI INT
               Fondo : INT
             FINE RECORD
```




```
/* funzione ADT Crea ( ) */
FUNZIONE Crea ( ) : LISTA
ℓ: LISTA
i: INT
INIZIO
PER i ← 1 A MAXLEN ESEGUI
  ℓ.Corpo[i] ← 0
  i ← i + 1
FINE PER
ℓ.Fondo ← 0
RITORNA (ℓ)
FINE
```

L_1 = [11, 3, 9, -7, 22, 4|




```
/* funzione ADT TestVuota ( ) */
FUNZIONE TestVuota (VAL ℓ: LISTA) : BOOL
esito: BOOL
INIZIO
SE (ℓ.Fondo = 0)
  ALLORA
    esito ← VERO
  ALTRIMENTI
    esito ← FALSO
FINE SE
RITORNA (esito)
FINE
```

```
/* funzione di servizio StampaLista ( ) */
PROCEDURA StampaLista (VAL ℓ: LISTA)
i: INT
INIZIO
SE (TestVuota (ℓ) = FALSO)
  ALLORA
    PER i ← 1 A ℓ.Fondo ESEGUI
      Scrivi(ℓ.Corpo[i])
      i ← i + 1
    FINE PER
    Scrivi(ℓ.Fondo)
  ALTRIMENTI
    Scrivi ("LISTA vuota!")
FINE SE
RITORNA
FINE
```

```
/* funzione ADT InsTesta ( ) */
PROCEDURA InsTesta (VAL ele: INT, REF ℓ: LISTA)
INIZIO
SE (ℓ.Fondo < MAXLEN)
  ALLORA                          //shift totale a dx
    PER i ← (ℓ.Fondo + 1) INDIETRO A 2 ESEGUI
      ℓ.Corpo[i] ← ℓ.Corpo[i - 1]
      i ← i – 1
    FINE PER
    ℓ.Corpo[1] ← ele              //insert elemento in testa
    ℓ.Fondo ← ℓ.Fondo + 1
  ALTRIMENTI
    Scrivi ("LISTA piena!")
FINE SE
RITORNA
FINE
```

```
/* funzione ADT InsFondo ( ) */
PROCEDURA InsFondo (VAL ele: INT, REF ℓ: LISTA)
INIZIO
SE (ℓ.Fondo < MAXLEN)
  ALLORA                          //shift totale a dx
    ℓ.Corpo[ℓ.Fondo + 1] ← ele    //insert elemento in fondo
    ℓ.Fondo ← ℓ.Fondo + 1
  ALTRIMENTI
    Scrivi ("LISTA piena!")
FINE SE
RITORNA
FINE
```

```
/* funzione ADT CancTesta ( ) */
FUNZIONE CancTesta (REF ℓ: LISTA, REF ele: INT) : BOOL
esito : BOOL
i : INT
INIZIO
SE (TestVuota (ℓ) = FALSO)
  ALLORA
    ele ← ℓ.Corpo[1]
     //shift totale a sx
    PER i ← 1 A (ℓ.Fondo – 1) ESEGUI
      ℓ.Corpo[i] ← ℓ.Corpo[i + 1]
      i ← i + 1
    FINE PER
    ℓ.Fondo ← ℓ.Fondo – 1
    esito ← VERO
  ALTRIMENTI
    Scrivi ("LISTA vuota!")
    esito ← FALSO
FINE SE
RITORNA (esito)
FINE
```

```
/* funzione ADT InsPos ( ) */
PROCEDURA InsPos (VAL ele: INT, VAL pos: INT, REF ℓ: LISTA)
INIZIO
SE (ℓ.Fondo < MAXLEN)
  ALLORA                          // shift parziale a dx
    SE (pos > 1 ) AND (pos < ℓ.Fondo)
      ALLORA
        PER i ← (ℓ.Fondo + 1) INDIETRO A (pos + 1) ESEGUI
          ℓ.Corpo[i] ← ℓ.Corpo[i - 1]
          i ← i – 1
        FINE PER
        ℓ.Corpo[pos] ← ele          //insert elemento in posizione pos
        ℓ.Fondo ← ℓ.Fondo + 1
      ALTRIMENTI
        Scrivi ("Operazione richiesta non possibile!")
    FINE SE
  ALTRIMENTI
    Scrivi ("LISTA piena!")
FINE SE
RITORNA
FINE
```

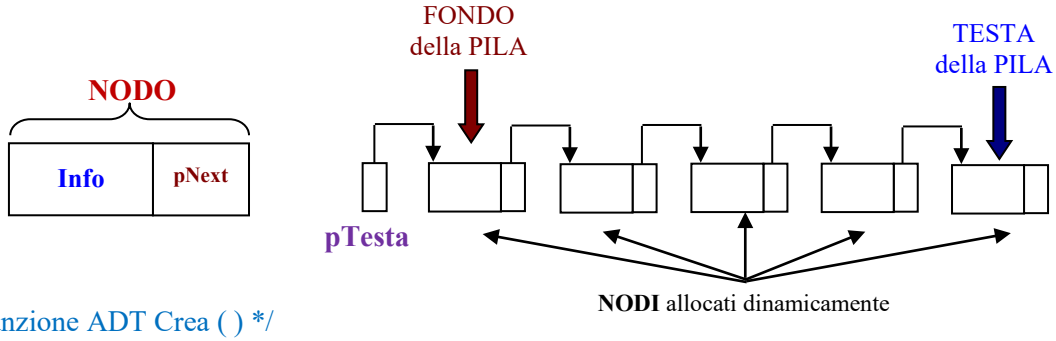
```
/* funzione ADT CancPos ( ) */
FUNZIONE CancPos (REF ℓ: LISTA, REF ele: INT, VAL pos: INT) : BOOL
esito : BOOL
i : INT
INIZIO
SE (TestVuota (ℓ) = FALSO)
  ALLORA
    SE (pos > 1) AND (pos < ℓ.fondo)
      ALLORA
        ele ← ℓ.Corpo[pos]
       //shift parziale a sx
       PER i ← pos A ℓ.Fondo ESEGUI
         ℓ.Corpo[i] ← ℓ.Corpo[i + 1]
         i ← i + 1
       FINE PER
       ℓ.Fondo ← ℓ.Fondo – 1
       esito ← VERO
    ALTRIMENTI
        Scrivi ("Operazione richiesta non possibile!")
        esito ← FALSO
    FINE SE
  ALTRIMENTI
    Scrivi ("LISTA vuota!")
    esito ← FALSO
FINE SE
RITORNA (esito)
FINE
```

```
/* funzione ADT CancFondo ( ) */
FUNZIONE CancFondo (REF ℓ: LISTA, REF ele: INT) : BOOL
esito : BOOL
INIZIO
SE (TestVuota (ℓ) = FALSO)
  ALLORA
    ele ← ℓ.Corpo[ℓ.Fondo]
    ℓ.Fondo ← ℓ.Fondo – 1
    esito ← VERO
  ALTRIMENTI
    Scrivi ("LISTA vuota!")
    esito ← FALSO
FINE SE
RITORNA (esito)
FINE
```

STRUTTURA DATI ASTRATTA LINEARE PILA

IMPLEMENTATA CON UNA STRUTTURA DATI NON SEQUENZIALE, AD ALLOCAZIONE DINAMICA ED AD ACCESSO NON SEQUENZIALE (OSSIA CON UNA LISTA LINKATA)

TIPO NODO = **RECORD**
Info : INT
pNext : PUNTATORE A NODO
FINE RECORD




```
/* funzione ADT Crea ( ) */
FUNZIONE Crea ( ): PUNTATORE A NODO
pTesta : PUNTATORE A NODO
INIZIO
pTesta ← NULL
RITORNA (pTesta)
FINE
```

```
/* funzione ADT TestVuota ( ) */
FUNZIONE TestVuota (VAL pTesta: PUNTATORE A NODO) : BOOL
esito: BOOL
INIZIO
SE (pTesta = NULL)
  ALLORA
    esito ← VERO
  ALTRIMENTI
    esito ← FALSO
FINE SE
RITORNA (esito)
FINE
```

```
/* funzione di servizio StampaPila ( ) */
PROCEDURA StampaPila (VAL pTesta: PUNTATORE A NODO)
pCurr: PUNTATORE A NODO
INIZIO
SE (TestVuota(pTesta) = FALSO)
  ALLORA
    pCurr ← pTesta
    MENTRE (pCurr ≠ NULL) ESEGUI
      Scrivi(pCurr→Info)
      pCurr ← (pCurr→pNext)
    FINE MENTRE
  ALTRIMENTI
    Scrivi ("PILA vuota!")
FINE SE
RITORNA
FINE
```

```
// oppure in alternativa si può scrivere
    pCurr ← pTesta
    MENTRE (pCurr ≠ NULL) ESEGUI
      Scrivi ((*pCurr).Info)
      pCurr ← ((*pCurr).pNext)
    FINE MENTRE
```

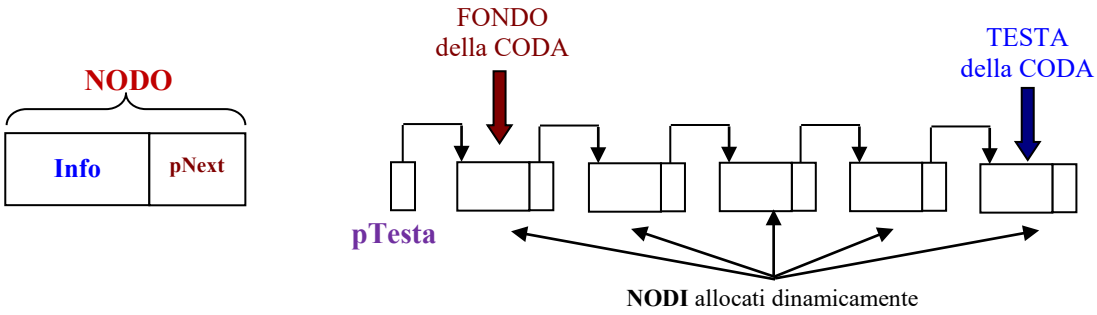
```
/* funzione di servizio DeallocaPila ( ) */
PROCEDURA DeallocaPila (REF pTesta: PUNTATORE A NODO)
pCurr: PUNTATORE A NODO
INIZIO
SE (TestVuota(pTesta) = FALSO)
  ALLORA
    pCurr ← pTesta
    MENTRE (pTesta ≠ NULL) ESEGUI
      pTesta ← (pCurr→pNext)
      Dealloca (pCurr)
      pCurr ← pTesta
    FINE MENTRE
    pTesta ← NULL              //importantissimo per dire che non c'è più la lista linkata
  ALTRIMENTI
    Scrivi ("PILA vuota!")
FINE SE
RITORNA
FINE

/* funzione ADT Push ( ) */
PROCEDURA Push (REF pTesta: PUNTATORE A NODO, VAL ele: INT)
pNew: PUNTATORE A NODO
pCurr: PUNTATORE A NODO
INIZIO
Alloca (pNew, DimensioneDi (NODO))
SE (pNew ≠ NULL)
  ALLORA
   //valorizzo i campi del nodo appena trovato
    (pNew→Info) ← ele
    (pNew→pNext) ← NULL
    SE (TestVuota(pTesta) = VERO)
      ALLORA
        //è il primo nodo in assoluto
        pTesta ← pNew
      ALTRIMENTI
        //non è il primo nodo della lista linkata…vado alla fine
        pCurr ← pTesta
        MENTRE (pCurr→pNext ≠ NULL) ESEGUI
          pCurr ← (pCurr→pNext)
        FINE MENTRE
       //importantissimo…… lego il nodo alla fine della lista linkata
        (pCurr→pNext) ← pNew
     FINE SE
  ALTRIMENTI
    Scrivi ("Errore di allocazione!")
FINE SE
RITORNA
FINE
```

```
/* funzione ADT Pop ( ) */
FUNZIONE Pop (REF pTesta: PUNTATORE A NODO, REF ele: INT): BOOL
pLast: PUNTATORE A NODO
pCurr: PUNTATORE A NODO
INIZIO
SE (TestVuota (pTesta) = VERO)
  ALLORA
    pCurr ← pTesta
    MENTRE (pCurr→pNext ≠ NULL) ESEGUI
      //penultimo nodo corrente
      pLast ← pCurr
      //ultimo nodo corrente
      pCurr ← (pCurr→pNext)
    FINE MENTRE
    // leggo il valore del nodo posto alla fine della lista linkata
     ele ← (pCurr→Info)
    // se c'è rimasto un solo nodo devo ripristinare la condizione di pila creata
     SE (pTesta = pCurr)
        ALLORA
          pTesta ← NULL
       ALTRIMENTI
         // metto il tappo al penultimo nodo della lista linkata (ora è l'ultimo della PILA)
         (pLast→pNext) ← NULL
    FINE SE
    // dealloco lo spazio di memoria dinamico dell'ultimo nodo della lista linkata (ultimo nodo della PILA)
     Dealloca (pCurr)
     esito ← VERO
  ALTRIMENTI
    Scrivi ("PILA vuota!")
     esito ← FALSO
FINE SE
RITORNA
FINE
```

STRUTTURA DATI ASTRATTA LINEARE CODA

IMPLEMENTATA CON UNA STRUTTURA DATI NON SEQUENZIALE, AD ALLOCAZIONE DINAMICA ED AD ACCESSO NON SEQUENZIALE (OSSIA CON UNA LISTA LINKATA)




```
/* funzione ADT Crea ( ) */
FUNZIONE Crea ( ) : PUNTATORE A NODO
pTesta: PUNTATORE A NODO
INIZIO
pTesta ← NULL
RITORNA (pTesta)
FINE
```

```
/* funzione ADT TestVuota ( ) */
FUNZIONE TestVuota (VAL pTesta: PUNTATORE A NODO) : BOOL
esito: BOOL
INIZIO
SE (pTesta = NULL)
  ALLORA
    esito ← VERO
  ALTRIMENTI
    esito ← FALSO
FINE SE
RITORNA (esito)
FINE
```

```
/* funzione di servizio StampaCoda( ) */
PROCEDURA StampaCoda (VAL pTesta: PUNTATORE A NODO)
pCurr: PUNTATORE A NODO
INIZIO
SE (TestVuota(pTesta) = FALSO)
  ALLORA
    pCurr ← pTesta
    MENTRE (pCurr ≠ NULL) ESEGUI
      Scrivi(pCurr→Info)
      pCurr ← (pCurr→pNext)
    FINE MENTRE
  ALTRIMENTI
    Scrivi ("CODA vuota!")
FINE SE
RITORNA
FINE
```

```
// oppure in alternativa si può scrivere
    pCurr ← pTesta
    MENTRE (pCurr ≠ NULL) ESEGUI
      Scrivi ((*pCurr).Info)
      pCurr ← ((*pCurr).pNext)
    FINE MENTRE
```

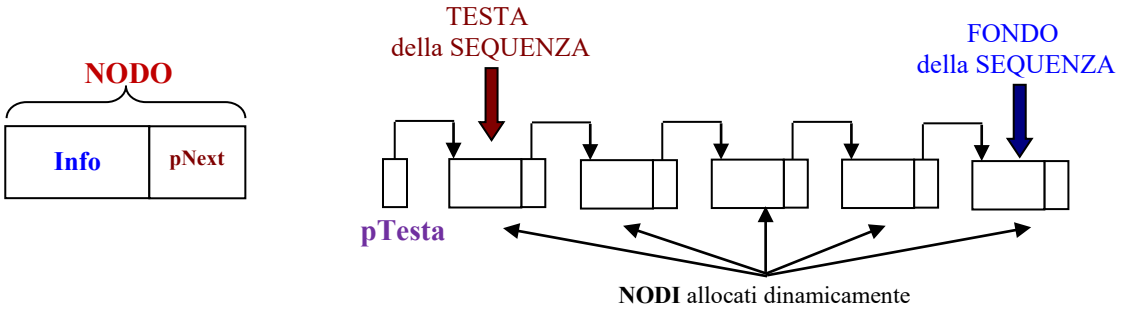
```
/* funzione di servizio DeallocaCoda ( ) */
PROCEDURA DeallocaCoda (REF pTesta: PUNTATORE A NODO)
pCurr: PUNTATORE A NODO
INIZIO
SE (TestVuota(pTesta) = FALSO)
  ALLORA
    pCurr ← pTesta
    MENTRE (pTesta ≠ NULL) ESEGUI
      pTesta ← (pCurr→pNext)
      Dealloca (pCurr)
      pCurr ← pTesta
    FINE MENTRE
    pTesta ← NULL              //importantissimo per dire che non c'è più la lista linkata
  ALTRIMENTI
    Scrivi ("CODA vuota!")
FINE SE
RITORNA
FINE

/* funzione ADT Inserisci ( ) */
PROCEDURA Inserisci (REF pTesta: PUNTATORE A NODO, VAL ele: INT)
pNew, pCurr: PUNTATORE A NODO
INIZIO
Alloca (pNew, DimensioneDi (NODO))
SE (pNew ≠ NULL)
  ALLORA
   // valorizzo il nodo creato
    (pNew→Info) ← ele
    // verifico se si tratta del primo nodo inserito
    SE (TestVuota (pTesta) = VERO)
      ALLORA
        (pNew→pNext) ← NULL
      ALTRIMENTI
        (pNew→pNext) ← pTesta
     FINE SE
     //importantissimo.. aggiorno il puntatore alla testa della lista linkata
     pTesta ← pNew
  ALTRIMENTI
    Scrivi ("Errore di allocazione!")
FINE SE
RITORNA
FINE
```

```
/* funzione ADT Estrai ( ) */
FUNZIONE Estrai (REF pTesta: PUNTATORE A NODO, REF ele: INT): BOOL
pLast: PUNTATORE A NODO
pCurr: PUNTATORE A NODO
INIZIO
SE (TestVuota (pTesta) = FALSO)
  ALLORA
    pCurr ← pTesta
   //se la CODA ha più di un nodo mi posiziono su penultimo ed ultimo nodo
    SE (pCurr → pNext ≠ NULL)
      ALLORA
        MENTRE (pCurr→pNext ≠ NULL) ESEGUI
          //penultimo nodo corrente
          pLast ← pCurr
         //ultimo nodo corrente
          pCurr ← (pCurr→pNext)
        FINE MENTRE
        // metto il tappo al penultimo nodo della lista linkata (ora è l'ultimo della CODA)
       (pLast→pNext) ← NULL
     //se la coda ha solo un nodo non dimentico di reinizializzare il puntatore alla testa
      ALTRIMENTI
        pTesta ← NULL
    FINE SE
    // leggo il valore del nodo posto alla fine della lista linkata
    ele ← (pCurr→Info)
    // dealloco lo spazio di memoria dinamico dell'ultimo nodo della lista linkata (ultimo nodo della CODA)
    Dealloca (pCurr)
    esito ← VERO
  ALTRIMENTI
    Scrivi ("PILA vuota!")
     esito ← FALSO
FINE SE
RITORNA
FINE
```

STRUTTURA DATI ASTRATTA LINEARE LISTA o SEQUENZA

IMPLEMENTATA CON UNA STRUTTURA DATI NON SEQUENZIALE, AD ALLOCAZIONE DINAMICA ED AD ACCESSO NON SEQUENZIALE (OSSIA CON UNA LISTA LINKATA)




```
/* funzione ADT Crea ( ) */
FUNZIONE Crea ( ) : PUNTATORE A NODO
pTesta : PUNTATORE A NODO
INIZIO
pTesta ← NULL
RITORNA (pTesta)
FINE
```

```
/* funzione ADT TestVuota ( ) */
FUNZIONE TestVuota (VAL pTesta: PUNTATORE A NODO) : BOOL
esito: BOOL
INIZIO
SE (pTesta = NULL)
  ALLORA
    esito ← VERO
  ALTRIMENTI
    esito ← FALSO
FINE SE
RITORNA (esito)
FINE
```

```
/* funzione di servizio StampaLista( ) */
PROCEDURA StampaLista (VAL pTesta: PUNTATORE A NODO)
pCurr: PUNTATORE A NODO
INIZIO
SE (TestVuota(pTesta) = FALSO)
  ALLORA
    pCurr ← pTesta
    MENTRE (pCurr ≠ NULL) ESEGUI
      Scrivi(pCurr→Info)
      pCurr ← (pCurr→pNext)
    FINE MENTRE
  ALTRIMENTI
    Scrivi ("LISTA vuota!")
FINE SE
RITORNA
FINE
```

```
// oppure in alternativa si può scrivere
    pCurr ← pTesta
    MENTRE (pCurr ≠ NULL) ESEGUI
      Scrivi ((*pCurr).Info)
      pCurr ← ((*pCurr).pNext)
    FINE MENTRE
```

```
/* funzione di servizio DeallocaLista ( ) */
PROCEDURA DeallocaCoda (REF pTesta: PUNTATORE A NODO)
pCurr: PUNTATORE A NODO
INIZIO
SE (TestVuota(pTesta) = FALSO)
  ALLORA
    pCurr ← pTesta
    MENTRE (pTesta ≠ NULL) ESEGUI
      pTesta ← (pCurr→pNext)
      Dealloca (pCurr)
      pCurr ← pTesta
    FINE MENTRE
    pTesta ← NULL              //importantissimo per dire che non c'è più la lista linkata
  ALTRIMENTI
    Scrivi ("LISTA vuota!")
FINE SE
RITORNA
FINE

/* funzione ADT InsTesta ( ) */
PROCEDURA InsTesta (REF pTesta: PUNTATORE A NODO, VAL ele: INT)
pNew, pCurr: PUNTATORE A NODO
INIZIO
Alloca (pNew, DimensioneDi (NODO))
SE (pNew ≠ NULL)
  ALLORA
   // valorizzo il nodo creato
    (pNew→Info) ← ele
    // verifico se si tratta del primo nodo inserito
    SE (TestVuota (pTesta) = VERO)
      ALLORA
        (pNew→pNext) ← NULL
      ALTRIMENTI
        (pNew→pNext) ← pTesta
     FINE SE
     //importantissimo…. aggiorno il puntatore alla testa della lista linkata
     pTesta ← pNew
  ALTRIMENTI
    Scrivi ("Errore di allocazione!")
FINE SE
RITORNA
FINE
```

```
/* funzione ADT InsFondo ( ) */
PROCEDURA InsFondo (REF pTesta: PUNTATORE A NODO, VAL ele: INT)
pNew: PUNTATORE A NODO
pCurr: PUNTATORE A NODO
INIZIO
Alloca (pNew, DimensioneDi (NODO))
SE (pNew ≠ NULL)
  ALLORA
    //valorizzo i campi del nodo appena trovato
    (pNew→Info) ← ele
    (pNew→pNext) ← NULL
    SE (TestVuota(pTesta) = VERO)
      ALLORA
        //è il primo nodo in assoluto
        pTesta ← pNew
      ALTRIMENTI
        //non è il primo nodo della lista linkata…vado alla fine
        pCurr ← pTesta
        MENTRE (pCurr→pNext ≠ NULL) ESEGUI
          pCurr ← (pCurr→pNext)
        FINE MENTRE
       //importantissimo…… lego il nodo alla fine della lista linkata
        (pCurr→pNext) ← pNew
    FINE SE
  ALTRIMENTI
    Scrivi ("Errore di allocazione!")
FINE SE
RITORNA
FINE


/* funzione di servizio MisuraLista( ) */
FUNZIONE MisuraLista (VAL pTesta: PUNTATORE A NODO): INT
pCurr: PUNTATORE A NODO
n: INT
INIZIO
n ← 0
SE (TestVuota(pTesta) = FALSO)
  ALLORA
    pCurr ← pTesta;
    MENTRE (pCurr != NULL) ESEGUI
      n ← n + 1
      pCurr ← (pCurr → pNext)
    FINE MENTRE
RITORNA (n)
FINE
```

```
/* funzione ADT InsPos ( ) */
PROCEDURA InsPos (REF pTesta: PUNTATORE A NODO, VAL ele: INT, VAL pos: INT)
pNew, pCurr, pLast: PUNTATORE A NODO
n, posCurr: INT
//calcolo quanti nodi ha la lista
n ← MisuraLista(pTesta)
//check se operazione InsPos è possibile
SE (pos > 1) AND (pos < n)
  ALLORA
    //alloco il nuovo nodo
    Alloca (pNew, DimensioneDi (NODO))
     SE (pNew ≠ NULL)
        ALLORA
       //valorizzo l'elemento inserito
       (pNew→Info) ← ele
       //scorro la lista fino a pos
       posCurr ← 1
       pCurr ← pTesta
      MENTRE (posCurr < pos) ESEGUI
          posCurr ← posCurr +1
          //alla fine nodo in posizione pos
          pLast ← pCurr
         //alla fine nodo dopo posizione pos
         pCurr ← (pCurr→pNext)
       FINE MENTRE
       //aggancio il nodo dopo pLast
       (pLast → pNext) ← pNew
       //ma prima di pCurr
       (pNew→pNext) ← pCurr
    ALTRIMENTI
      Scrivi ("Errore di allocazione!")
  FINE SE
ALTRIMENTI
  Scrivi ("Operazione non ammessa")
FINE SE
RITORNA
FINE
```

```
/* funzione ADT CancFondo ( ) */
FUNZIONE CancFondo (REF pTesta: PUNTATORE A NODO, REF ele: INT): BOOL
pLast, pCurr: PUNTATORE A NODO
esito:BOOL
INIZIO
SE (TestVuota (pTesta) = VERO)
  ALLORA
    pCurr ← pTesta
    MENTRE (pCurr→pNext ≠ NULL) ESEGUI
      //penultimo nodo corrente
      pLast ← pCurr
      //ultimo nodo corrente
      pCurr ← (pCurr→pNext)
    FINE MENTRE
    ele ← (pCurr→Info)
   // dealloco lo spazio di memoria dinamico dell'ultimo nodo della lista linkata (ultimo nodo della LISTA)
    Dealloca (pCurr)
    // metto il tappo al penultimo nodo della lista linkata (ora è l'ultimo della LISTA)
    (pLast→pNext) ← NULL
    esito ← VERO
  ALTRIMENTI
    Scrivi ("LISTA vuota!")
    esito ← FALSO
FINE SE
RITORNA (esito)
FINE

/* funzione ADT CancTesta ( ) */
FUNZIONE CancTesta (REF pTesta: PUNTATORE A NODO, REF ele: INT): BOOL
pCurr: PUNTATORE A NODO
esito: BOOL
INIZIO
SE (TestVuota (pTesta) = VERO)
  ALLORA
    //restituisco il valore del primo nodo della lista
    ele ← (pTesta→Info)
    //inizializzo il ptr corrente alla testa della lista linkata
    pCurr ←pTesta
    //controllo se la lista ha almeno un altro nodo
    SE (pCurr → pNext != NULL)
      ALLORA
        //aggiorno il puntatore alla testa della lista linkata (2° nodo)
        pTesta ← (pCurr → pNext)
      ALTRIMENTI
        //se ho cancellato l'ultimo devo reinizializzare a NULL pTesta
        pTesta ← NULL
    FINE SE
    //libero lo spazio allocato per il nodo eliminato
    Dealloca(pCurr)
    esito ← VERO
  ALTRIMENTI
    Scrivi ("LISTA vuota!")
    esito ← FALSO
FINE SE
RITORNA (esito)
FINE
```

```
/* funzione ADT CancPos ( ) */
FUNZIONE CancPos (REF pTesta: PUNTATORE A NODO, VAL pos: INT, REF ele: INT): BOOL
pPrev, pPos, pSucc: PUNTATORE A NODO
n, posCurr: INT
esito: BOOL
//controllo se posso prelevare ancora un nodo nella LISTA
SE (TestVuota(pTesta) == FALSO)
  ALLORA
    //check su numero nodi della lista
    n ← MisuraLista(pTesta);
    SE ((pos > 1) && (pos < n))
      ALLORA
        // scorro tutta la lista linkata fino alla posizione pos
        posCurr ← 1
        pPrev ← pTesta
        pPos ← pTesta
        MENTRE (posCurr < pos) ESEGUI
          posCurr ← posCurr + 1
          pPrev ← pPos                  //nodo in posizione prima di pos
          pPos ← (pPrev -> pNext)       //nodo in posizione pos
          pSucc ← (pPos -> pNext)       //nodo in posizione dopo pos
        FINE MENTRE
        //restituisco il valore
        ele ← (pPos → Info)
        //aggancio il nodo dopo pPos
       (pPrev → pNext) ← pSucc
       // dealloco pPos
       Dealloca(pPos)
       //valorizzo esito
       esito ← VERO
      ALTRIMENTI
        Scrivi("Operazione CancPos del nodo non consentita con la posizione inserita!")
        esito ← FALSO
    FINE SE
  ALTRIMENTI
    Scrivi("LISTA vuota!")
    esito ←FALSO
FINE SE
RITORNA (esito)
FINE
```