

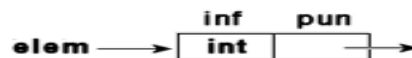
LE LISTE

Le liste sono uno strumento potente e versatile per la gestione dinamica della memoria, ma anche molto fragile e complesso. Il problema da risolvere nella gestione dinamica della memoria è quello di **definire una struttura dati** che sia **"effettivamente dinamica"**, cioè che permetta la modifica, **in fase di esecuzione del programma**, del numero degli elementi che la costituiscono (inserimento e cancellazione **dinamica** di un qualsiasi elemento **in qualsiasi momento ed in qualsiasi posizione**) **e che, contemporaneamente, mantenga un numero di "puntatori statici" costanti e indipendente dal numero di elementi dinamici aggiunti**.

Apparentemente questo problema sembrerebbe privo di soluzione in quanto, per poter fare ciò, dovremmo dotare ogni **variabile dinamica di una variabile di tipo puntatore per indirizzarla, dovrebbero esserci tante variabili quanti sono i puntatori**, quindi **se le variabili sono in numero non noto anche il numero dei puntatori non può essere noto a priori (quanti "puntatori statici" dobbiamo dichiarare all'inizio?)**.

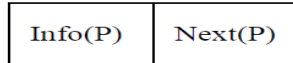
Come capiremo nel corso di questo modulo, l'uso delle liste risolve questo problema e offre quella dinamicità di cui si può avere bisogno durante lo sviluppo di un programma.

Una lista non è altro che una collezione di elementi omogenei, ma, **a differenza dell'array, ogni elemento della lista occupa in memoria una posizione qualsiasi**, che tra l'altro **può cambiare dinamicamente durante l'utilizzo della lista** stessa, inoltre **la dimensione della lista non è nota a priori e può variare nel tempo** (l'opposto dell'array, in cui la dimensione è ben nota e non è modificabile, memoria statica). Una lista può contenere uno o più campi contenenti informazioni, e, **necessariamente, deve contenere un puntatore per mezzo del quale è possibile il collegamento all'elemento successivo**.



Per allocare dinamicamente la memoria necessaria ad un elemento della lista, usiamo la funzione **malloc()** insieme a **sizeof**. **La lista base ha un solo campo informazione ed un puntatore**, come mostrato di seguito:

Elemento = informazione + puntatore

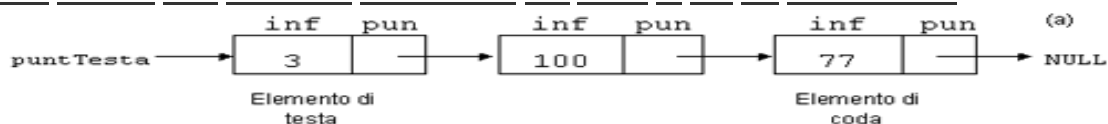


tradotto in codice, abbiamo:

```
struct elemento
{
    int inf;
    struct elemento *pun;
}
```

Una particolarità delle liste (a differenza, ad esempio, degli array) è la presenza all'interno della struttura dei **puntatori**.

Come si è detto, una lista può contenere **uno o più campi di informazione**, che possono essere di tipo int (come nell'esempio di sopra), char, float, ecc.; mentre **deve esserci sempre un campo che punta ad un altro elemento della lista**, che è **NULL se non ci sono altri elementi successivi da puntare, mentre risulta essere una struttura elementoP nel caso vi sia un successore**.



Per dichiarare una lista, basta scrivere (riferendosi alla struttura dichiarata precedentemente):

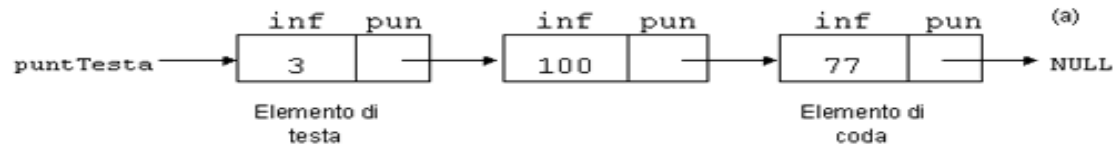
```
struct elemento *lista;
```

che altro non è che un puntatore ad una struttura elemento, e come tale potrebbe essere inizializzata anche ad un valore NULL, che identificherebbe una lista vuota. In questo modo definiamo, quindi, una **lista lineare**, ovvero una lista che deve essere visitata (o scandita) in ordine, cioè dal primo elemento fino all'ultimo, che viene identificato perché punta a NULL. Da ricordare che, anche se nella sua **rappresentazione logica**, una lista risulta essere sequenziale, **in realtà l'allocazione di**


memoria relativamente agli elementi è libera, cioè ogni volta che devo aggiungere un elemento alla lista, devo allocare la memoria relativa (e per fare questo utilizzo la funzione malloc), connetterlo all'ultimo elemento ed inserirvi l'informazione. La rappresentazione grafica di un elemento della lista è, come visto precedentemente, la seguente:



mentre quella di una lista risulta essere come segue:



Introduciamo alcune definizioni:



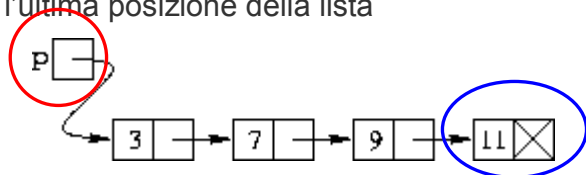
LISTA SEMPLICE

Una **lista semplice** è una sequenza di elementi aventi le seguenti caratteristiche:

- ▶ tutti gli elementi sono omogenei, ossia appartengono allo stesso tipo;
- ▶ in ogni elemento è presente un solo campo di tipo puntatore;
- ▶ ogni elemento è contraddistinto da una posizione.

In base alla posizione di ciascun elemento nella lista è possibile individuare le relazioni di precedenza e successione tra gli elementi. Tra tutti gli elementi, due hanno particolare interesse:


- **Elemento di testa**: occupa la prima posizione della lista
- **Elemento di coda**: occupa l'ultima posizione della lista



Tranne quelli di testa e di coda, gli altri elementi della lista sono definiti:

- **Elemento predecessore**, che occupa la posizione precedente ad un certo elemento
- **Elemento successivo**, che occupa la posizione successiva ad un certo elemento

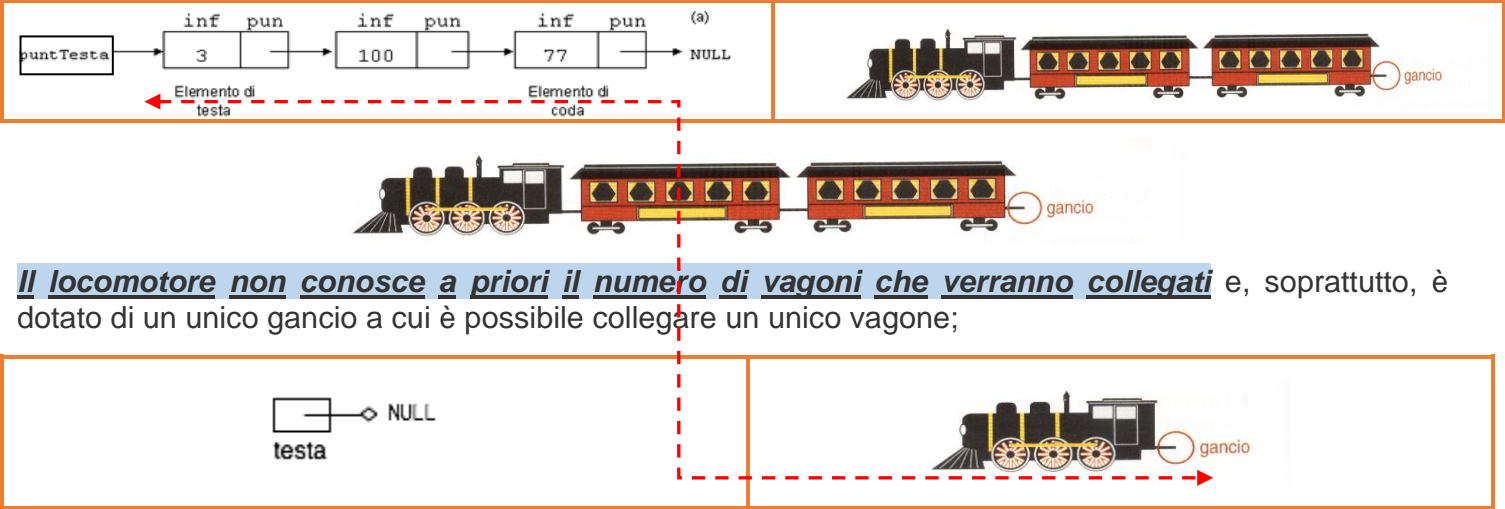
Come è evidente, il primo elemento della lista non ha predecessore e l'ultimo elemento non ha successore.



STATO DI UNA LISTA SEMPLICE

- ▶ Una **lista** si dice **vuota** se non contiene alcun elemento.
- ▶ Una **lista** si dice **unitaria** se è costituita da un solo elemento: in questo caso l'elemento di testa e l'elemento di coda coincidono.
- ▶ Una **lista** si dice **ordinata** se per tutti gli elementi vale una regola di inserimento basata sul valore di un determinato campo dell'elemento stesso.

Per capire meglio suggeriamo la seguente analogia “ferroviaria”: una locomotiva che traina una serie di vagoni ferroviari, all'interno dei quali ci sono delle persone (le informazioni).



Il locomotore non conosce a priori il numero di vagoni che verranno collegati e, soprattutto, è dotato di un unico gancio a cui è possibile collegare un unico vagon;

per poter aggiungere un secondo vagone, **è necessario disporre di un gancio libero che, sappiamo, è presente sul vagone che viene collegato.**



Il risultato è quello che segue, che ci ripresenta la situazione di partenza, **abbiamo un gancio libero** a cui possiamo collegare un solo altro vagone.



La stessa situazione si ripropone aggiungendo un secondo vagone e così via, rendendo quindi teoricamente infinito il numero di vagoni collegabili ad un unico locomotore.



Definiamo la corrispondenza con la struttura dati dinamica: il “locomotore informatico” è il puntatore statico, mentre il “vagone” è un record in cui un campo è un **puntatore**, al quale sarà possibile collegare altri record (al limite, in numero infinito).

Descriviamo una struttura (un record) che simuli l’esempio del treno

```
1 struct s_vagone
2 {
3     int info;           // campo informazione
4     struct s_vagone *gancio; // campo per il collegamento
5 };
6 typedef struct s_vagone vagone;
7 typedef vagone *gancio; // puntatore ad un vagone
```

Il record appena definito è costituito da un unico campo **info**, contenente un’informazione numerica; in tutta la trattazione teorica verranno presi in considerazione record simili a questo senza timore di perdere di generalità, in quanto si vuole focalizzare l’attenzione sul problema, già di sua natura complesso, della manipolazione dei puntatori senza introdurre elementi di “dispersione della concentrazione”.

Quando un record viene collegato al puntatore, quest’ultimo viene si occupato, ma **la funzione malloc(vagone)** che ha creato il record con un campo puntatore automaticamente mette a disposizione un nuovo puntatore libero; **in questo modo non è necessario conoscere, a priori, il numero di record per definire i puntatori necessari alla gestione della lista, perché provvede a questa la stessa istruzione malloc(vagone).**

Vediamo come creare due vagoni e collegarli tra di loro, iniziando dalla definizione delle variabili statiche

```
22 // definizione delle variabili statiche e loro inizializzazione |
23 int tempo;
24 gancio pLocomotore;           /* puntatore al primo elemento della lista */
25 gancio pCarrozza;             /* puntatore al vagone che verrà allocato */
26 tempo=0;                     /* inizializzazione variabile intera */
27 pLocomotore= NULL;           /* inizializzazione dei puntatori */
28 pCarrozza = NULL;
```

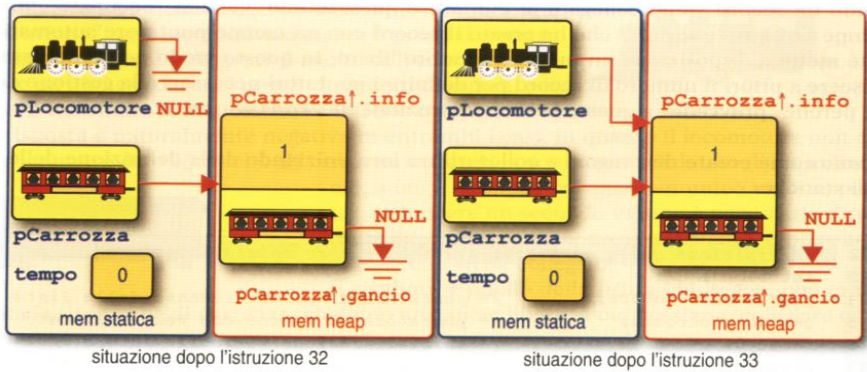
Sono state definite tre variabili, **pLocomotore**, **pCarrozza** e **tempo** (puntatore temporaneo) che sono memorizzate nella **memoria statica** e inizializzate a zero. La situazione della memoria dopo l’esecuzione dell’ultima istruzione (la 28), è illustrata nella seguente figura



Il passo successivo è quello di **creare**, dopo la prima variabile di tipo dinamico, un vagone e di collegarlo al locomotore; nel campo info viene scritto, a titolo informativo, un numero che indica la posizione della carrozza all'interno del convoglio.

Per capire meglio, vediamo passo passo la costruzione di un treno con due "vagoni"

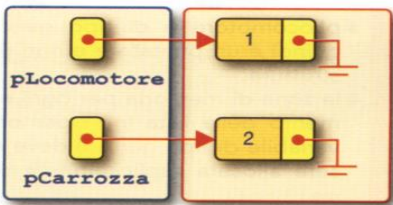
```
29 // creazione del primo vagone e collegamento al locomotore
30 pCarrozza = malloc(sizeof(vagone)); /* allocazione primo elemento */
31 pCarrozza->info = 1;                /* dati nel primo elemento */
32 pCarrozza->gancio = NULL;           /* gancio nel primo elemento */
33 pLocomotore=pCarrozza;              /* collego il locomotore */
```



Per aggiungere un nuovo vagone, dapprima si crea nella memoria heap una nuova variabile referenziata dal puntatore statico **pCarrozza**, utilizzando l'istruzione malloc(), quindi vengono inizializzati i due campi record e successivamente si collega il vagone al resto del treno, cioè il secondo elemento alla lista.

```
34 // creazione del secondo vagone
35 pCarrozza = malloc(sizeof(vagone)); /* allocazione secondo elemento */
36 pCarrozza->info = 2;                /* dati nel secondo elemento */
37 pCarrozza->gancio = NULL;           /* gancio nel secondo elemento */
```

Le prime operazioni sono identiche a quelle viste per la creazione del primo vagone ma le istruzioni che realizzano il collegamento del vagone al resto del treno non sono immediate. La situazione delle variabili a questo punto della costruzione è quella schematizzata sotto

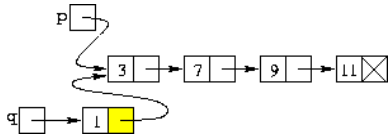
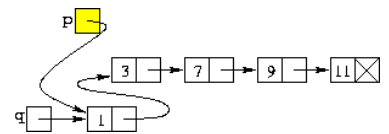


in cui abbiamo sostituito i disegni del locomotore e dei vagoni con le variabili puntatore. L'inserimento di un nuovo elemento, come vedremo successivamente, può essere effettuato in due posizioni (in testa o in coda).

INSERIMENTO DI UN ELEMENTO IN TESTA AD UNA LISTA

Inserire un nuovo elemento in testa ad una lista significa **posizionarlo come primo elemento** della lista, cioè collegarlo al puntatore di testa **pLocomotore**. Guardiamo la tabella di sotto per capire meglio:

A		Creazione del nuovo nodo (in giallo) con l'aiuto di una variabile d'appoggio q
		Scrittura del dato (campo informazione) nel nuovo nodo

B		Concatenazione del nuovo nodo in testa alla lista
C		Dirottamento del puntatore "ufficiale" della lista al nuovo elemento

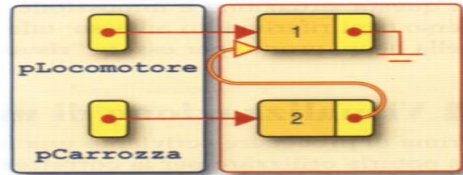
Tale operazione non può esaurirsi in una sola istruzione in quanto si potrebbe perdere il resto del treno. Le operazioni da eseguire, guardiamo sopra, sono:

- Creare un nuovo vagone
- Collegare il nuovo vagone al resto del treno
- Staccare il locomotore e collegarlo al nuovo convoglio

Se non viene rispettato l'ordine di questa sequenza, per esempio scambiando la fase B con la C, si perde il collegamento al vagone di testa preesistente.
Dopo aver creato un nuovo vagone lo colleghiamo al resto del convoglio; l'istruzione che effettua il collegamento è la seguente:

pCarrozza → gancio = pLocomotore

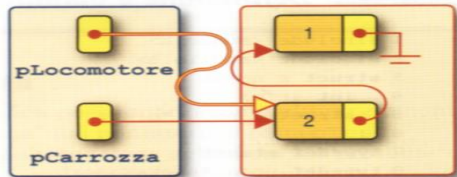
L'indirizzo del primo elemento è contenuto nel puntatore pLocomotore, quindi assegno al gancio del secondo vagone l'indirizzo del primo vagone.



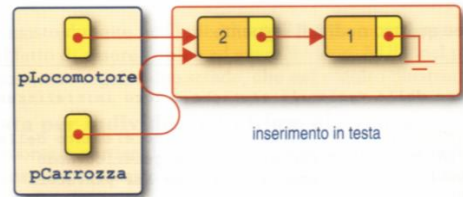
In questa situazione il vagone di testa preesistente viene puntato da due puntatori, quindi è possibile scollegare uno dei due puntatori senza perderne l'indirizzamento. Con l'istruzione

pLocomotore = pCarrozza

distacco il locomotore dalla vecchia testa cioè il primo vagone e lo collego alla nuova testa attuando il collegamento al nuovo convoglio. In questa situazione il nuovo vagone viene puntato da due puntatori, quindi è possibile liberare il puntatore pCarrozza per referenziare nuovi vagoni



la situazione finale è quella di sotto che è identica a quella di prima ma con i due vagoni allineati



Prima di andare avanti, nell'esempio che segue vediamo il programma in C per la creazione di una lista di testa e la sua visualizzazione

```
#include <stdio.h>
#include <stdlib.h>

struct s_nodo
{
    int info;           // campo informazione
    struct s_nodo *next; // campo per il collegamento
}
```

```

};
typedef struct s_nodo nodo; // nodo della lista
typedef nodo *puntaNodo; // puntatore ad un elemento

void stampaLista(puntaNodo lista)
{
    while (lista != NULL)
    {
        printf("%4d ", lista->info);
        lista = lista->next;
    }
}; /* stampaLista */

main()
{
    // definizione delle variabili statiche e loro inizializzazione
    int x;
    puntaNodo pTesta; /* puntatore al primo elemento della lista */
    puntaNodo pNode; /* puntatore ad un generico nodo */
    pTesta = NULL; /* inizializzazione dei puntatori */
    pNode = NULL;

    // creazione di una lista di testa

    for (x=0; x<3; x++)
    {
        pNode = malloc(sizeof(nodo)); /* allocazione nuovo elemento */
        pNode->info = x+1; /* scrivo le informazioni */
        pNode->next = pTesta; /* collego all'ultimo inserito */
        pTesta = pNode; /* collego il locomotore */
    }
    printf("\n\n Visualizzo la lista\n");
    stampaLista(pTesta); /* visualizza la lista */

    // creo direttamente e il primo elemento sul locomotore
    pTesta = malloc(sizeof(nodo)); /* allocazione primo elemento */

    // lista numerica creata senza puntatori temporanei
    nodo lis; /* puntatore al primo elemento della lista */
    pTesta = malloc(sizeof(nodo)); /* allocazione primo elemento */
    pTesta->info = 1;

    pTesta->next = malloc(sizeof(nodo)); /* allocazione secondo elemento */
    pTesta->next->info = 2;

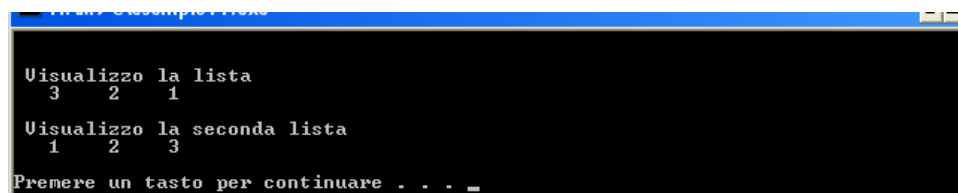
    pTesta->next->next = malloc(sizeof(nodo)); /* alloc. terzo elemento */
    pTesta->next->next->info = 3;
    pTesta->next->next->next = NULL;

    printf("\n\n Visualizzo la seconda lista\n");
    stampaLista(pTesta); /* visualizza la lista */

    printf("\n\n");
    system("PAUSE");
}

```

Mandando in esecuzione il programma (**l'inserimento è in testa**) otteniamo:



```

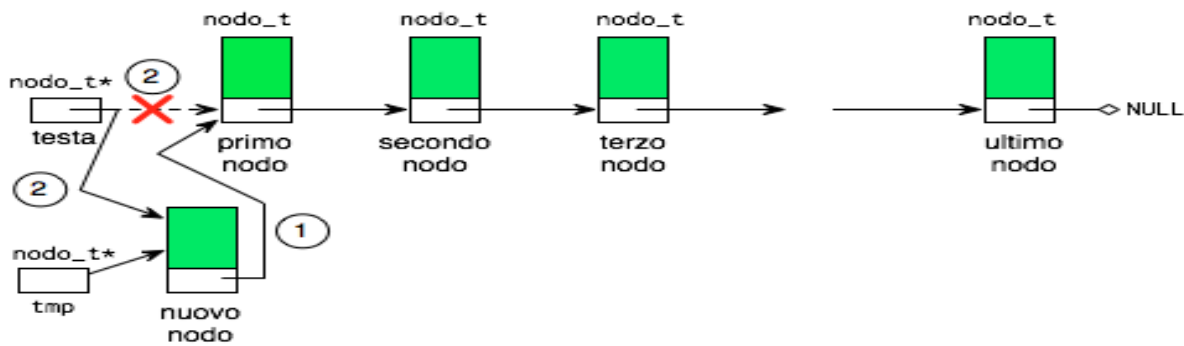
Visualizzo la lista
3    2    1

Visualizzo la seconda lista
1    2    3

Premere un tasto per continuare . . . _

```

Riassumendo per l'inserimento in testa



INSERIMENTO DI UN ELEMENTO IN CODA AD UNA LISTA

Inserire un nuovo elemento in coda ad una lista significa posizionarlo alla fine della lista, nella posizione più lontana dal puntatore di testa pTesta. Guardiamo la tabella di sotto per capire meglio:

	Ricerca dell'ultimo nodo della lista
	Creazione del nuovo nodo e assegnazione dell'indirizzo al puntatore che prima era NULL
	Inserimento dei dati nel nuovo nodo

Questa operazione non presenta particolari difficoltà qualora nella lista sia presente un solo elemento; ciò in quanto il puntatore dell'unico elemento presente nella lista, che è pTesta, viene fatto puntare al nuovo elemento, referenziato da pNodo, mentre risulta molto complesso nel caso in cui la lista contenga un numero non precisato di elementi.

Generalmente una lista contiene un numero indeterminato di elementi, proprio per la sua natura dinamica. Per poter collegare un elemento in coda, quindi dopo l'ultimo elemento presente, è necessario individuare l'indirizzo di memoria dell'ultimo elemento e scrivere nel suo puntatore l'indirizzo del nuovo nodo da collegare in coda, che diventerà il nuovo ultimo elemento.

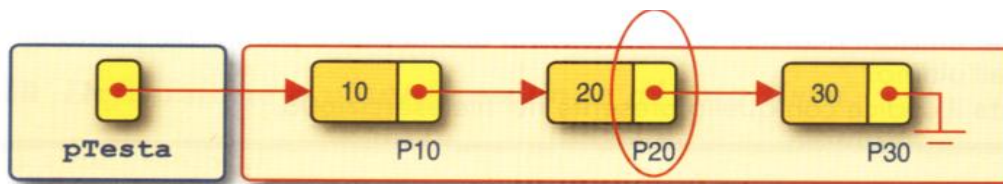
Il primo passo consiste dunque nell'individuare l'ultimo record (l'ultimo vagone) presente nella lista, che **è caratterizzato dal fatto di essere l'unico ad avere il valore NULL nel puntatore.**

Per individuare l'ultimo record, a partire dalla testa, scorriamo la lista fino a raggiungere la coda; non conoscendo il numero di elementi che compongono la lista, è necessario utilizzare un ciclo a condizione iniziale come quello impostato nella tabella che segue

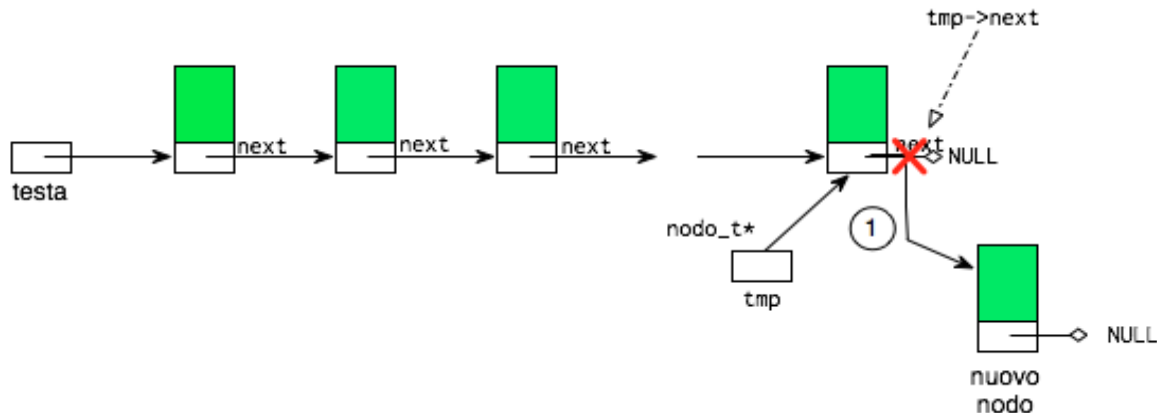
I Affinamento	II Affinamento
Analizza il primo elemento	pCorrente ← primo elemento della lista
Trova l'ultimo elemento della lista	<u>mentre</u> il puntatore è diverso da NULL passa ad analizzare il successivo elemento
Collega il nuovo elemento	Copia nel puntatore l'indirizzo del nuovo elemento

Questo è un tipico esempio di come sia "delicato" l'utilizzo delle liste a puntatori e di come sia facile commettere errori: per ridurre al minimo tale eventualità è consigliabile utilizzare disegni e modelli grafici del problema prima di stendere la codifica, verificando sempre le condizioni estreme di funzionamento (analogamente a quanto si fa nella progettazione dei cicli).

Riprendiamo il disegno della lista e procediamo con il progetto del codice "a ritroso", cioè a partire dal risultato ricerchiamo il modo per ottenerlo. Guardiamo la figura di sotto



Il nuovo elemento della lista, che vogliamo inserire alla fine della lista, deve essere collegato dopo l'elemento della lista con valore pari a 30 (è l'ultimo elemento della lista, infatti P30 = NULL); l'indirizzo di questo elemento è quello contenuto nel puntatore dell'elemento precedente, denominato P20. Una volta individuato questo elemento, il suo puntatore (P30 nel nostro caso) non sarà più NULL, ma assumerà il **valore dell'indirizzo** del nuovo elemento che voglio inserire (valore restituito precedentemente dalla funzione malloc) il puntatore di questo elemento successivamente verrà ad assumere il valore NULL, ad indicare che adesso è il nuovo elemento finale della lista. Riassumendo per l'inserimento in coda



Per capire meglio, guardiamo il seguente programma per la creazione di una lista di coda e sua visualizzazione

/* creazione di una lista di coda e sua visualizzazione */

```
#include <stdio.h>
#include <stdlib.h>
struct s_nodo
{
    int info;          // campo informazione
    struct s_nodo *next; // campo per il collegamento
};
typedef struct s_nodo nodo; // nodo della lista
typedef nodo *puntaNodo; // puntatore ad un elemento

void stampaLista(puntaNodo lista)
{
    while (lista != NULL)
    {
        printf("%4d ", lista->info);
        lista = lista->next;
    }
}; /* stampaLista */

void stampaListaRicorsiva(puntaNodo lista)
{
    if (lista != NULL)
    {
        printf("%4d ", lista->info);
        stampaListaRicorsiva(lista->next);
    }
} /* StampaListaRicorsiva */

void stampaListaInvertita(puntaNodo lista)
    /* Stampa la lista lis in ordine invertito. Versione ricorsiva. */
```



```

{
    if (lista != NULL)
    {
        stampaListaInvertita(lista->next);
        printf("%4d ", lista->info);
    }
} /* StampaListaInvertita */

puntaNodo creaListaTesta1(int quanti)
{
    int x;
    puntaNodo pNodo;
    puntaNodo pTesta;
    pTesta= NULL; /* inizializzazione dei puntatore iniziale */
    for (x=0; x<quanti; x++)
    {
        pNodo = malloc(sizeof(nodo)); /* allocazione nuovo elemento */
        pNodo->info = x+1; /* scrivo le informazioni */
        pNodo->next = pTesta; /* collego all'ultimo inserito */
        pTesta=pNodo; /* collego il locomotore */
    }
    return pTesta;
}; /* creaListaTesta */

void inserisciCodaLista(puntaNodo *lista, int dato)
{
    puntaNodo ultimo; /* puntatore usato per la scansione */
    puntaNodo pNodo; /* puntatore di servizio temporaneo */
    /* creazione del nuovo record */
    pNodo = malloc(sizeof(nodo));
    pNodo->info = dato;
    pNodo->next = NULL; /* NULL perchè sarà l'ultimo in lista */
    /* scansione della lista per trovare l'ultimo*/
    if (*lista == NULL)
        *lista = pNodo;
    else
    {
        ultimo = *lista;
        while (ultimo->next!= NULL)
            ultimo = ultimo->next;
    }
    /* concatenazione del nuovo record in coda alla lista */
    ultimo->next = pNodo;
} /* fine inserisciCodaLista */

main()
{
    // definizione delle variabili statiche e loro inizializzazione
    int x;
    puntaNodo pTesta; /* puntatore al primo elemento della lista */
    puntaNodo pNodo; /* puntatore ad un generico nodo */
    pTesta= NULL; /* inizializzazione dei puntatori */
    pNodo = NULL;
    // creazione di una lista di coda
    inserisciCodaLista(&pTesta,10); /* aggiungo un elemento in coda */
    inserisciCodaLista(&pTesta,20); /* aggiungo un elemento in coda */
    inserisciCodaLista(&pTesta,30); /* aggiungo un elemento in coda */
    printf("\n\n Visualizzo la lista ricorsivamente \n");
    stampaListaRicorsiva(pTesta); /* visualizza la lista ricorsiva */
    printf("\n\n Visualizzo la lista ricorsivamente ed invertita\n");
}

```

```

        stampaListaInvertita(pTesta);    /* visualizza la lista ricorsiva */
        printf("\n\n");
        system("PAUSE");
    }

```

Mandando in esecuzione il programma otteniamo

```

Visualizzo la lista ricorsivamente
10 20 30

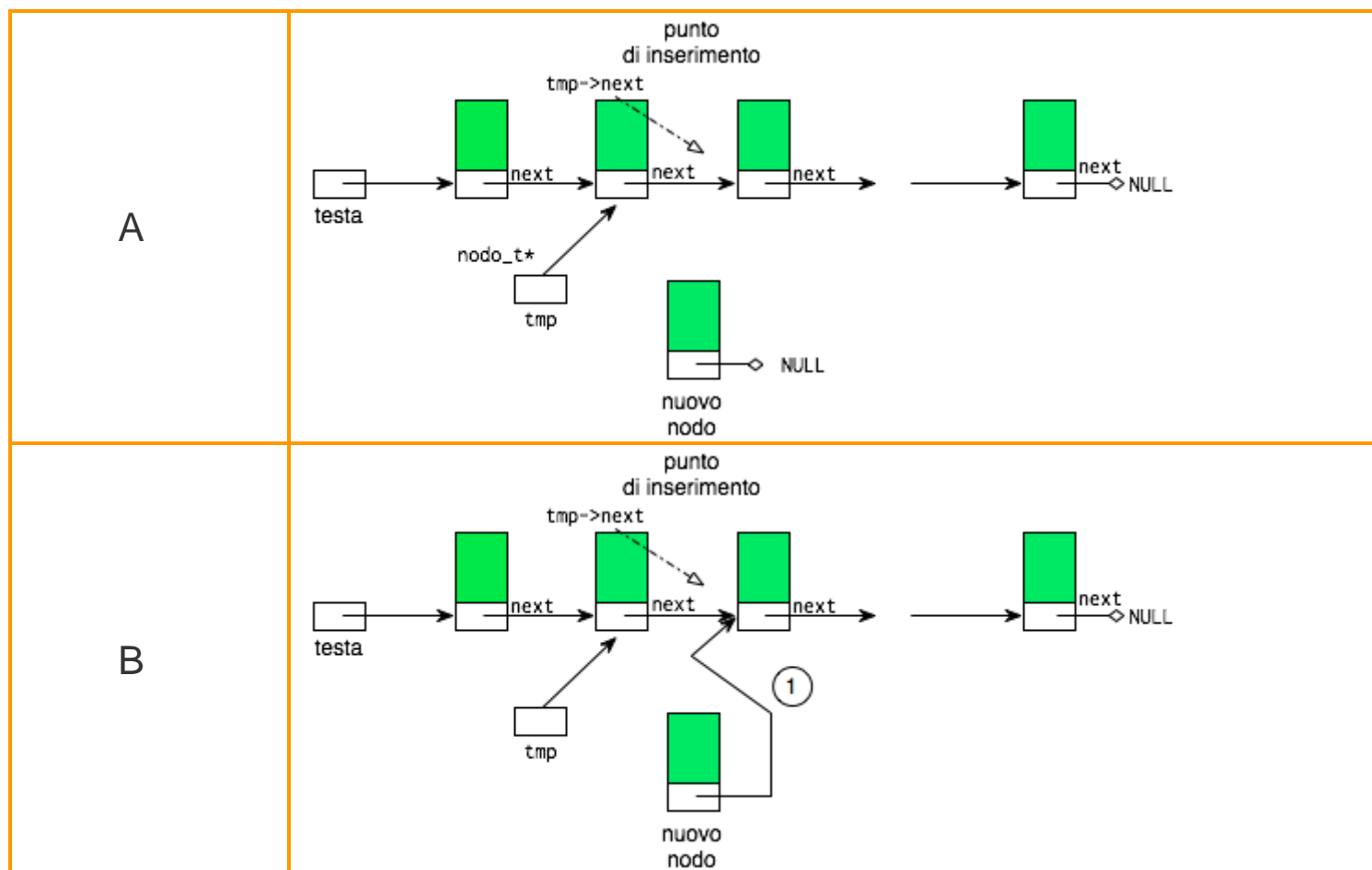
Visualizzo la lista ricorsivamente ed invertita
30 20 10

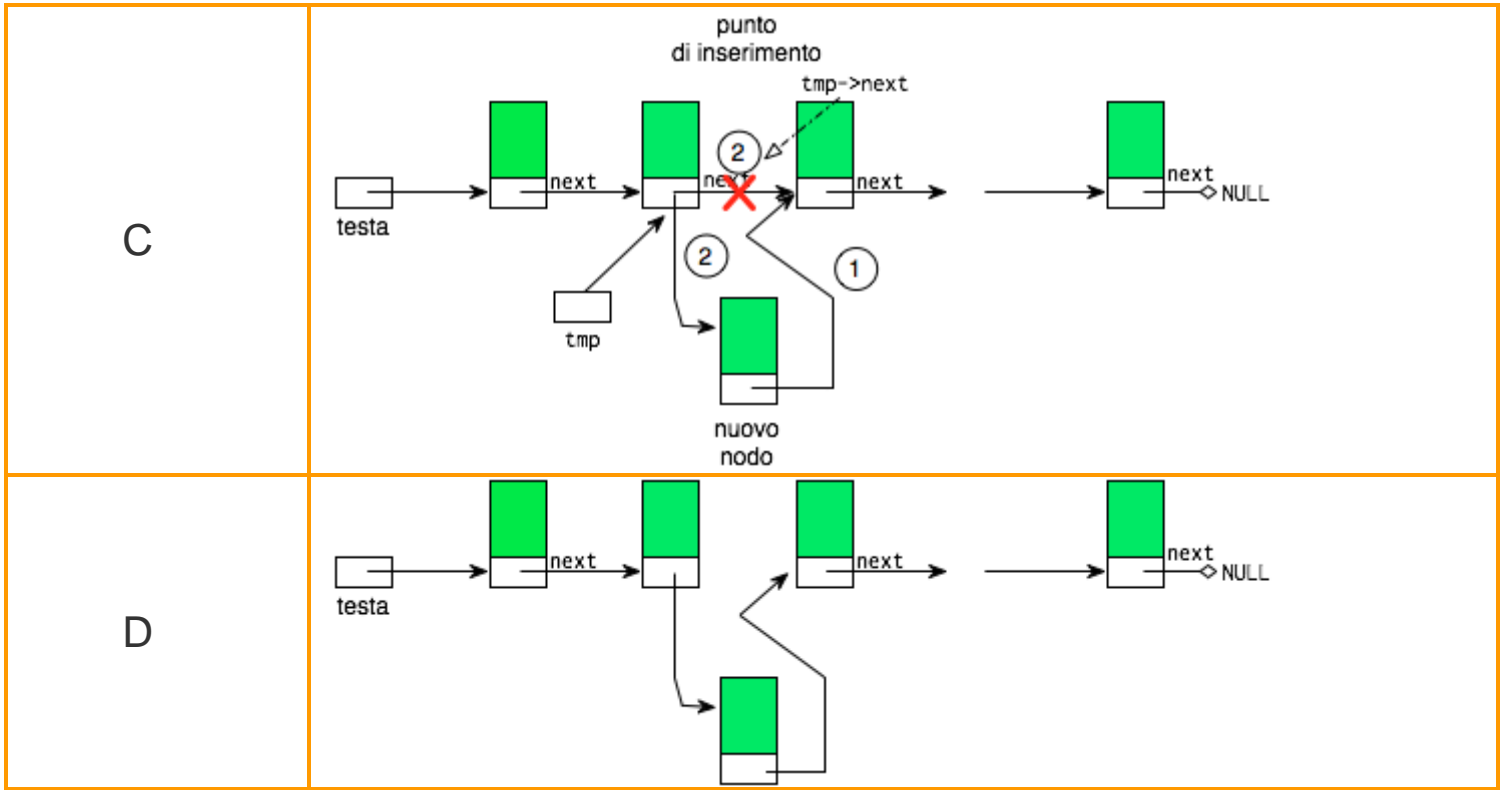
Premere un tasto per continuare . . .

```

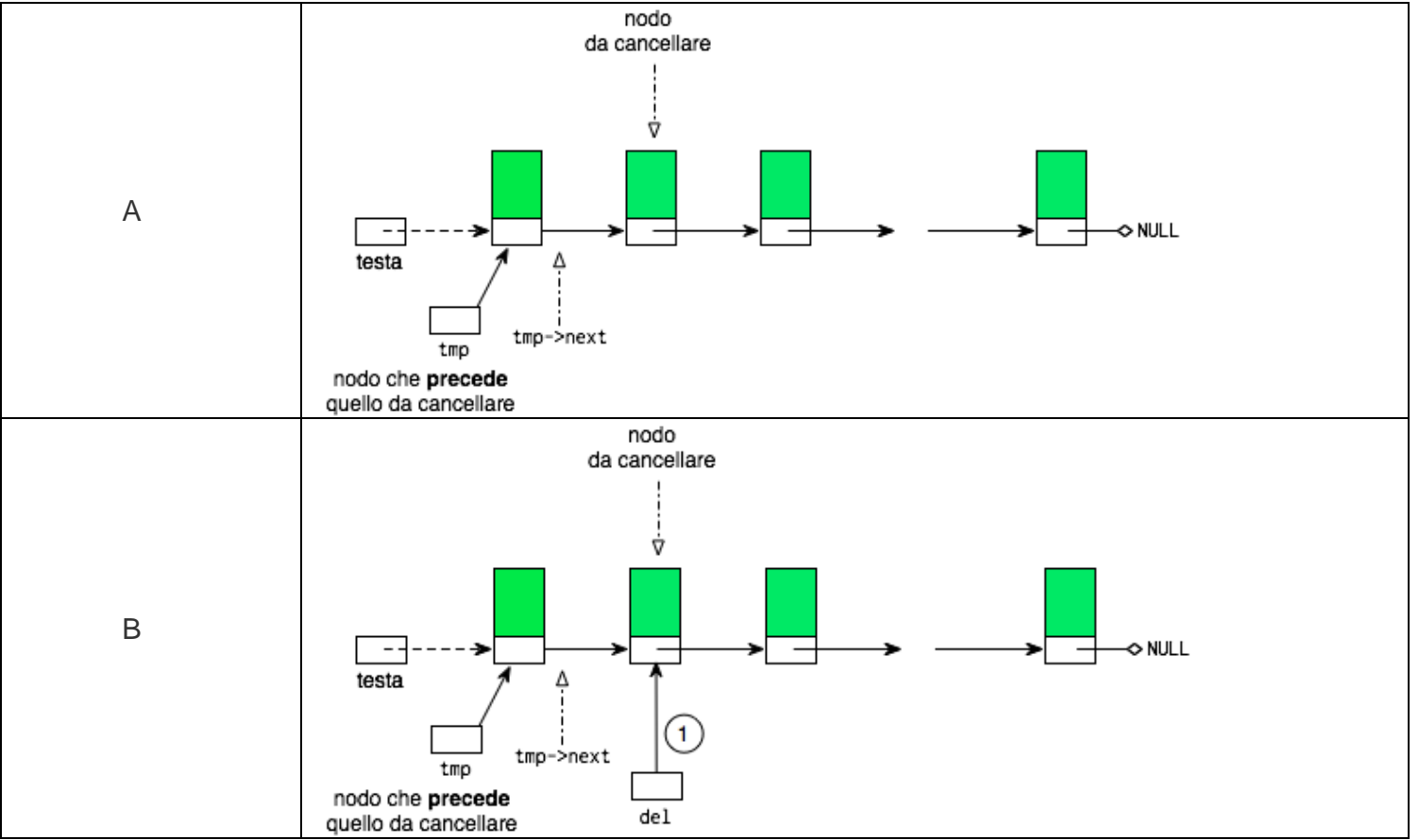
Può essere necessario inserire un nuovo elemento in un punto della lista che non è né la testa né la coda: in questo caso c'è una fase iniziale in cui si identifica il punto di inserimento, **individuando l'elemento che precede il punto di inserimento**, quindi si procede con l'aggiornamento dei puntatori. Poiché si cerca l'elemento precedente rispetto al punto di inserimento, è necessario gestire in modo adeguato il fatto che l'inserimento avvenga in testa, perché in tal caso l'elemento precedente non esiste.

La sequenza di modifiche dei puntatori è riportata nelle figure di sotto





Nella figura di sotto, viene invece rappresentata la sequenza di operazioni che dobbiamo fare per l'eliminazione di un elemento



C	<p>nodo da cancellare</p> <p>testa</p> <p>tmp</p> <p>tmp->next</p> <p>del</p> <p>nodo che precede quello da cancellare</p>
D	<p>nodo da cancellare</p> <p>testa</p> <p>tmp</p> <p>tmp->next</p> <p>del</p> <p>nodo che precede quello da cancellare</p>
E	<p>nodo da cancellare</p> <p>testa</p> <p>tmp</p> <p>tmp->next</p> <p>del</p> <p>nodo che precede quello da cancellare</p>

Il programma C per l’inserimento in un nuovo elemento in un punto qualsiasi della lista e per l’eliminazione di un elemento dalla lista viene lasciato come esercizio