


Laboratorio di Algoritmi e Strutture Dati I

Esercitazione 6

Test di laboratorio: parte 1

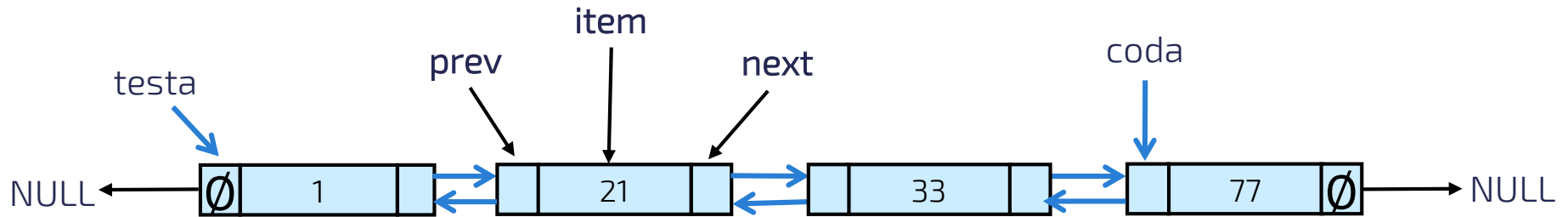
- Martedì 28: **1° Test** + Esercitazione 7
- Il Test comprenderà gli argomenti visti fino all'esercitazione 6 (inclusa).
- Non dovrete programmare: saranno domande a risposta multipla.
- Regole:
 - Il test dà diritto a max **1 punto bonus** (sul voto finale).
 - È facoltativo.
 - Non si può recuperare.
 - Si svolgerà su eLearning a partire dalle 11:00 e avrà la durata di 16 minuti.
- È obbligatorio iscriversi su eLearning (elearning.unica.it) nell'apposito form entro e non oltre venerdì 24 alle 12:00.

Lezione 6: Reminder sulle liste

- Insieme **dinamico** di elementi, ad accesso **sequenziale**;
- Ogni elemento ha almeno una **chiave** e un **riferimento** all'elemento successivo nella lista (nelle liste singolarmente concatenate).
- Diversi tipi di lista:
 - Lineari singolarmente concatenate
 - Lineari doppiamente concatenate 
 - Circolari singolarmente concatenate
 - Circolari doppiamente concatenate
- Le liste oggetto dell'esercitazione odierna hanno **due riferimenti**, uno all'elemento **precedente** e uno al **successivo**.

Lezione 6: Liste lineari doppiamente concatenate

- Ogni nodo della lista ha un campo **item**, più un riferimento (puntatore) al nodo **precedente** e uno al nodo **successivo**.



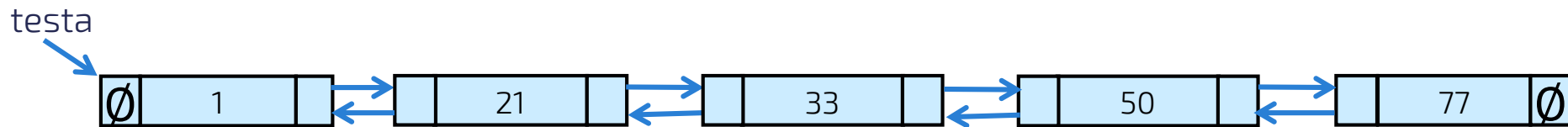
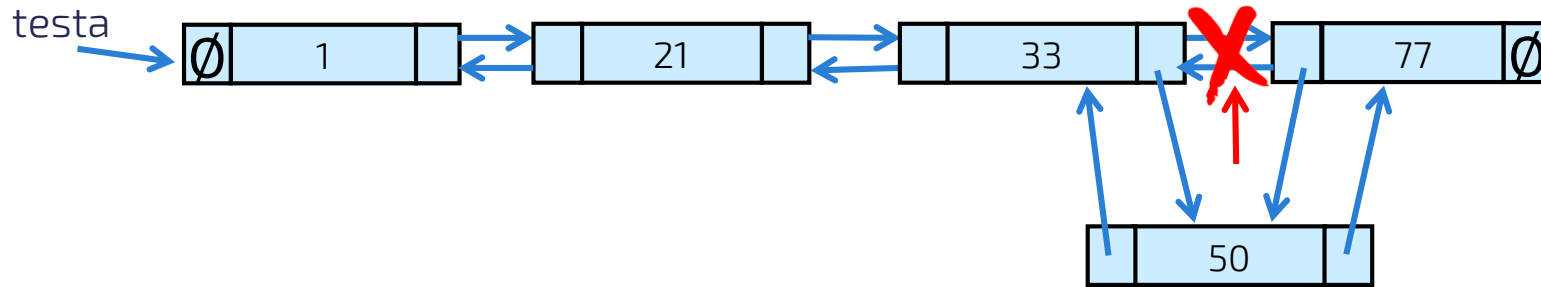
- Un generico nodo di una lista doppiamente concatenata può essere così definito:

```
struct nodo
{
    <type> item;
    struct nodo* prev;
    struct nodo* next;
};

typedef struct nodo Nodo;
```

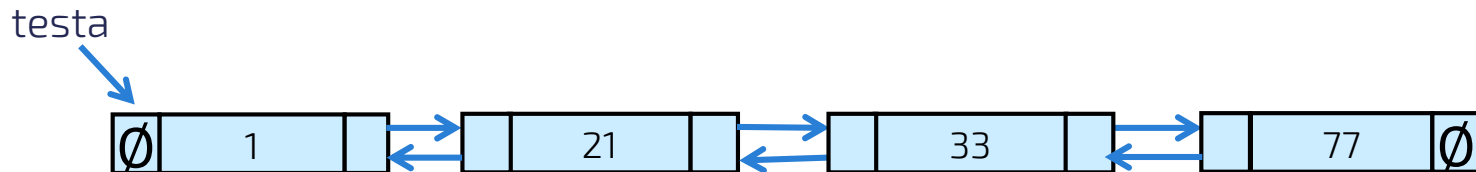
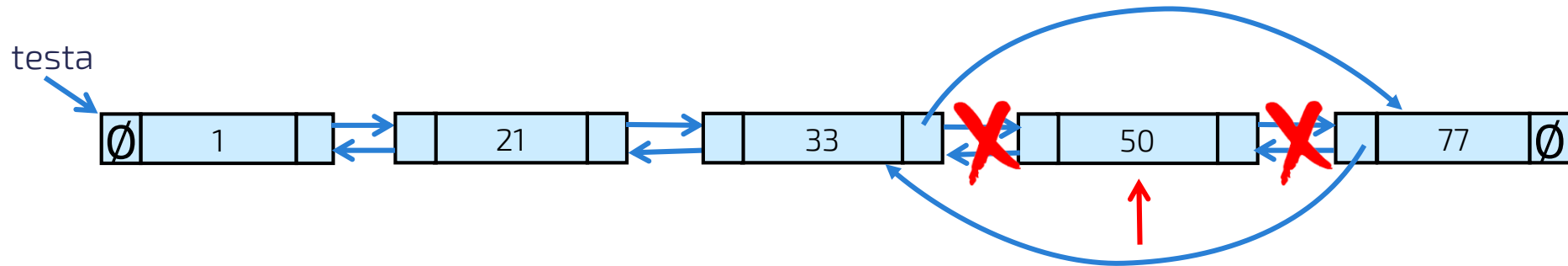
Lezione 6: LLDC Inserimento Ordinato

- Si cerca la posizione in cui inserire il nuovo elemento (in base all'ordine);
- Si inserisce il nuovo elemento adattando i dovuti collegamenti.



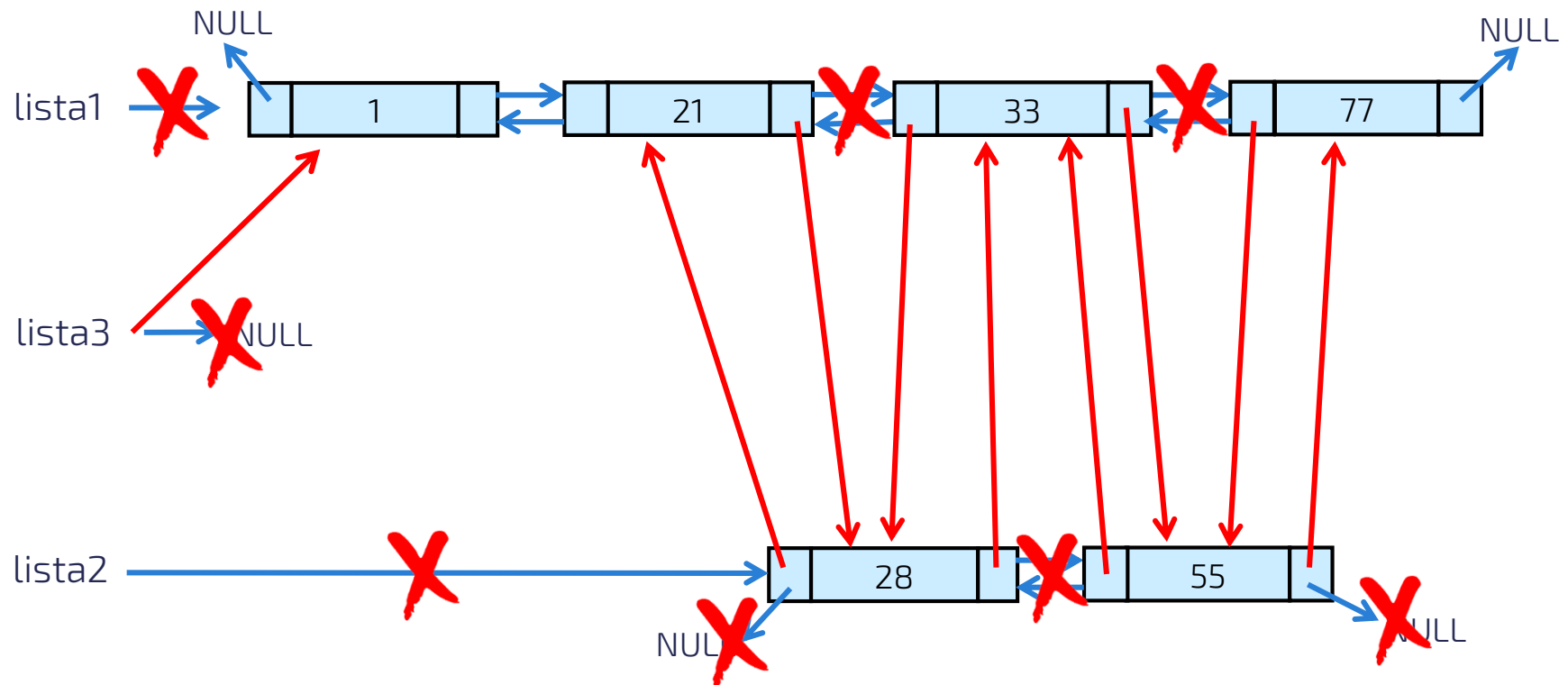
Lezione 6: LLDC Cancellazione

- Si cerca l'elemento da eliminare;
- Si “modificano” i collegamenti nella lista affinché l'elemento non faccia più parte della lista;
- Si dealloca l'elemento eliminato dalla lista.



Lezione 6: LLDC Fusione di due liste

- Si crea un nuovo "puntatore di testa";
- Si collegano i nodi delle liste originali (senza allocarne di nuovi) in modo tale da ottenere una nuova lista ordinata;
- Si sfrutta l'ordinamento delle due liste originali;
- Dopo la fusione la nuova lista ha dimensione $n+m$;
- Dopo la fusione le due liste iniziali hanno dimensione 0.



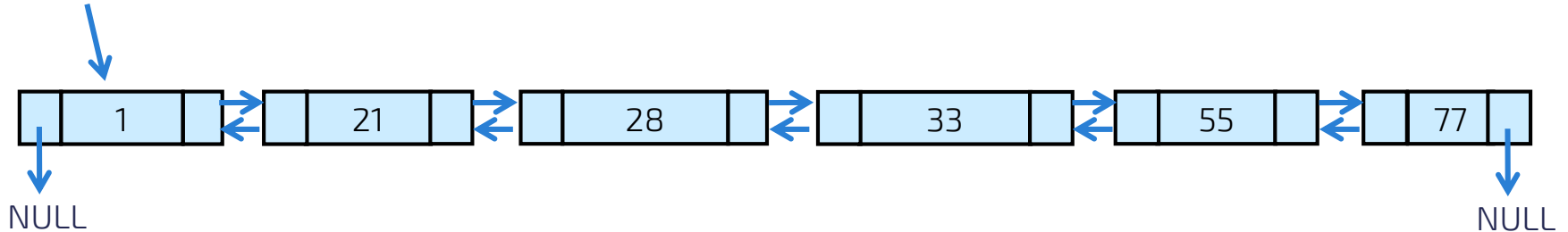
Lezione 6: Fusione di due liste

- Dopo la fusione:

lista1 → NULL

lista2 → NULL

lista3



- Le liste originali risultano entrambe vuote
- NESSUN nuovo nodo è stato allocato (sono stati sfruttati i nodi delle liste precedenti).

continue()...

Esercizio 6: Netflix o Spotify



Vogliamo rappresentare una versione semplificata di un catalogo NETFLIX o di un catalogo SPOTIFY tramite liste doppiamente concatenate.

- La lista rappresenterà una **playlist di serie TV (o di canzoni)**.
- Ogni nodo della lista rappresenterà una serie TV (o una canzone) con le seguenti informazioni:
 - Titolo della serie (o titolo della canzone);
 - Genere;
 - Numero di episodi della serie (o durata temporale della canzone).
- Gli elementi saranno **ordinati alfabeticamente** in base al titolo.
- Scegliete **uno solo** dei due progetti!

Laboratorio di Algoritmi e Strutture Dati I

Esercitazione 6

Netflix version

Esercizio 6 – Netflix version

NETFLIX

Vogliamo rappresentare una versione semplificata di un catalogo NETFLIX.

- La lista rappresenterà una **playlist di serie TV**.
- Ogni nodo della lista rappresenterà una serie TV con le seguenti informazioni:
 - Titolo della serie;
 - Genere;
 - Numero di episodi totali della serie.
- Gli elementi saranno **ordinati alfabeticamente** in base al titolo.

Esercizio 6: progetto base

- Scaricate da eLearning il progetto base per l'esercitazione odierna;
- Il file comprende le seguenti strutture:
 - TVS per rappresentare una nuova serie tv (nodo della lista);
 - Playlist per rappresentare un'intera playlist di serie tv (la lista);
- Il file comprende le seguenti funzioni (già implementate):
 - *acquireTVSeries()* \rightarrow *TVS** : chiede all'utente le informazioni riguardo una nuova serie tv, crea il nuovo nodo e lo restituisce come puntatore (pronto per essere inserito nella lista, nella posizione corretta);
 - *printTVSeries(TVS* t)* \rightarrow *void* : prende in input il puntatore ad un elemento della lista e stampa le informazioni in esso contenute;
- **NON** modificate le funzioni e le strutture già presenti!

Esercizio 6: Creazione e gestione PlayList

- Scrivere un algoritmo che simuli la gestione di una lista di **serie TV**.
- Implementare una lista **doppiamente concatenata** in modo che i nodi siano organizzati in ordine alfabetico sulla base del valore del campo dati di tipo stringa (**titolo della serie**).
- Le specifiche del problema sono le seguenti:
 - Ciascun dato della lista è costituito da un **titolo**, un **genere** ed un **numero di episodi**;
 - I dati sono memorizzati in ordine alfabetico (per titolo);
 - Deve poter essere inserita una nuova serie TV;
 - Deve poter essere eliminata (e deallocata) una serie TV già presente;
 - Un dato già inserito deve poter essere modificato (cosa succede se modifico il titolo?);
 - Deve essere possibile visualizzare i dati **singolarmente** oppure simulando uno **scrolling** che opera in senso crescente e decrescente (con delle funzioni **next** e **prev**);
 - Deve essere possibile **fondere** due playlist in una unica come spiegato precedentemente.

Esercizio 6: Le strutture

Gli elementi della lista sono costituiti da un **titolo** (stringa), da un **genere** (stringa) ed un **numero di episodi** (intero).

```
struct tvs
{
    char title[DIM_TITLE];
    char genre[DIM_GENRE];
    int num_episodes;
    struct tvs* prev;
    struct tvs* next;
};

typedef struct tvs TVS;
```

Una playlist (lista) è così definita:

```
struct playlist
{
    char name[DIM_NAME];
    TVS* top;
};

typedef struct playlist PlayList;
```

Esercizio 6: Le funzioni

Dovranno essere presenti le seguenti funzioni:

```
//chiede all'utente i dati relativi ad una nuova serie tv, alloca e  
riempie il nuovo nodo e ne restituisce il puntatore
```

```
algoritmo acquireTVSeries() → TVS*
```

Già implementata!

```
//inserisce una serie tv (creata con la funzione acquire) nella lista  
in maniera ordinata
```

```
algoritmo insertTVSeries(Playlist* pl, TVS* t) → void
```

```
//cerca una serie tv nella lista (tramite il titolo) e restituisce il  
suo puntatore
```

```
algoritmo findTVSeries(Playlist* pl, char title[]) → TVS*
```

```
//modifica una serie tv già presente (trovata tramite la funzione  
find) e la riposiziona nella posizione corretta
```

```
algoritmo modifyTVSeries(Playlist* pl, TVS* t) → void
```

```
//elimina una serie tv già presente (trovata tramite la funzione  
find) e dealloca l'elemento
```

```
algoritmo deleteTVSeries(Playlist* pl, TVS* t) → void
```


Esercizio 6: Le funzioni

Dovranno essere presenti le seguenti funzioni:

```
//stampa le informazioni relative ad una singola serie tv  
algoritmo printTVSeries(TVS* t) → void
```

Già implementata!

```
//stampa l'intera playlist di serie tv (usando la funzione di stampa  
singola)  
algoritmo printPlayList(PlayList* pl) → void
```

```
//restituisce il puntatore alla serie tv successiva rispetto a quella  
passata come puntatore se presente, altrimenti restituisce NULL  
algoritmo nextTVSeries(TVS* t) → TVS*
```

```
//restituisce il puntatore alla serie tv precedente rispetto a quella  
passata come puntatore se presente, altrimenti restituisce NULL  
algoritmo prevTVSeries(TVS* t) → TVS*
```

```
//fonde le liste plA e plB in un'unica lista plC (anch'essa passata come parametro). NESSUN nuovo  
nodo deve essere allocato  
algoritmo mergePlayList(PlayList* plA, PlayList* plB, PlayList* plC) → void
```

Esercizio 6: Inserimento ordinato (hint)

Pseudo codice dell'inserimento ordinato:

```
algoritmo insertTVSeries(Playlist* pl, TVS* t) → void

if(playlist vuota) then
    <gestire il caso per la lista vuota>
else
    //ricerca della posizione in cui inserire (tmp_prev e tmp_next sono due puntatori d'appoggio)
    tmp_prev = NULL
    tmp_next = testa della playlist
    while(tmp_next ≠ NULL and tmp_next->title ≤ t->title) //attenzione al confronto tra stringhe!
        tmp_prev = tmp_next
        tmp_next = tmp_next->next

    //ora conosciamo il nodo a cui appendere il nuovo nodo t
    if(siamo arrivati alla fine della playlist) then
        <gestire il caso di inserimento in coda>

    else if(dobbiamo inserire prima del primo elemento della playlist) then
        <gestire il caso di inserimento in testa>

    else
        <gestire il caso di inserimento generico tra due nodi>
```

Esercizio 6: strutture e puntatori (reminder)

- C'era una volta una struttura in C che aveva un campo *item* ed un campo *link*.

```
struct nodo
{
    int item;
    struct nodo *link;
}
typedef struct nodo Nodo;
```

- Se dichiaro una **struttura** accedo ai suoi campi come:

```
Nodo n = ...
n.item = ...
n.link = ...
```

- Se dichiaro un **puntatore a struttura** accedo ai suoi campi come:

```
Nodo *n = ...
n->item = ...
n->link = ...
```

- Nelle funzioni...
 - Se passo una struttura come parametro?
 - Se passo il puntatore ad una struttura come parametro?

error: member reference type 'Nodo *' (aka 'struct nodo *') is a pointer; maybe you meant to use '->'
error: member reference type 'Nodo' (aka 'struct nodo') is not a pointer; maybe you meant to use '!'?

end().