UNIVERSITÀ
DI TORINO

Laboratorio di Programmazione I

Lezione n. 7: Operazioni su array

Alessandro Mazzei

Slides: prof. Elvio Amparore

Outline

- Domandine
- Trasformazione elementi array
- Copia selettiva e riduzione sul posto
- modifica_array.c
- Lab07-Es1 Regole trasformazione array

- Aritmetica dei puntatori con le stringhe
- codifica.c
- Lab07-Es2 Elimina vocali

- Intervalli semiaperti
- my_string.c

- Lab07-Es3 Sottosequenze con errori

Domanda 1

Cosa definisce questo codice?

```
int *countPtr, count;
```

Soluzione

Cosa definisce questo codice?

```
int* countPtr, count;
```

- `countPtr` è una variabile puntatore a dati di tipo int;
- `count` è una variabile di tipo int.

Nessuna delle due variabili è inizializzata.
Per inizializzarle possiamo scrivere, ad esempio:

```
int* countPtr = NULL, count = 0;
```

Domanda 2

Cosa fa questo codice?

```
int count = 3;
int* pC = &count;
int* pC2 = &pC;
printf("%d %d\n", count, pC);
```

Soluzione

Cosa fa questo codice?

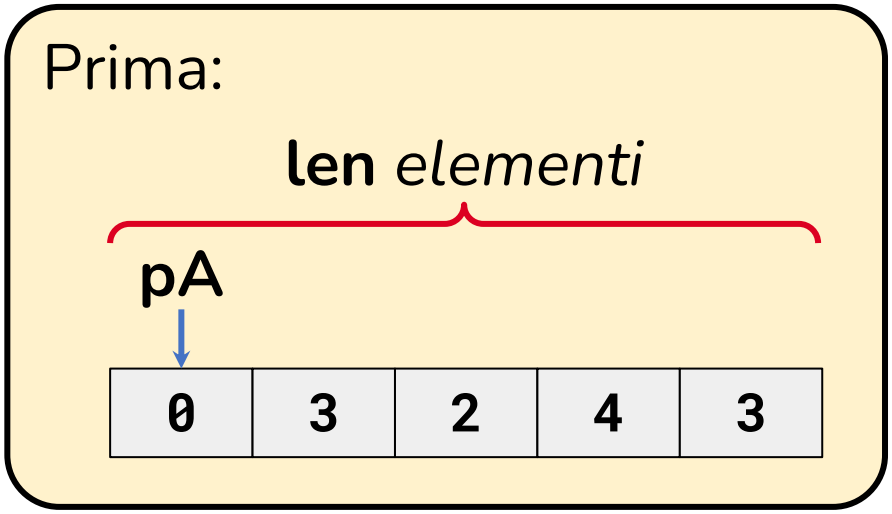
```
int count = 3;
int* pC = &count;
int* pC2 = &pC;
printf("%d %d\n", count, *pC);
```

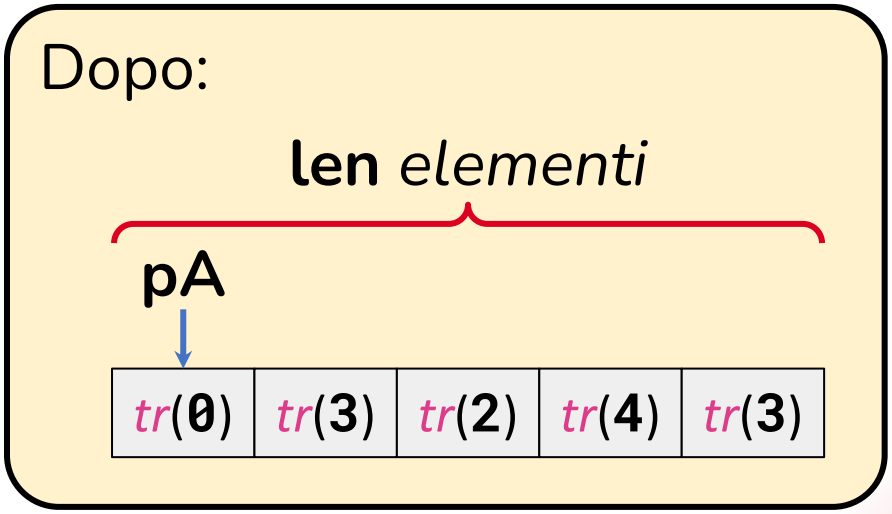
- &pC non è un puntatore a int (ma è un doppio puntatore!)
- Accedere ai valori tramite dereferenziamento richiede l'operatore *.

Trasformazione elementi array

Dato un puntatore **pA** ad una sequenza di dati di lunghezza **len**, vogliamo applicare una funzione di trasformazione *tr* elemento per elemento. Come fare?

```
for (size_t i=0; i<len; i++) {
    pA[i] = tr(pA[i]);
}
```





Copia selettiva elementi array: *filtri*

Dato un puntatore **pA** ad una sequenza di dati di lunghezza **lenA**, vogliamo copiare in un'altra sequenza **pB** tutti gli elementi che rispettano un criterio *Cond*. Come fare?

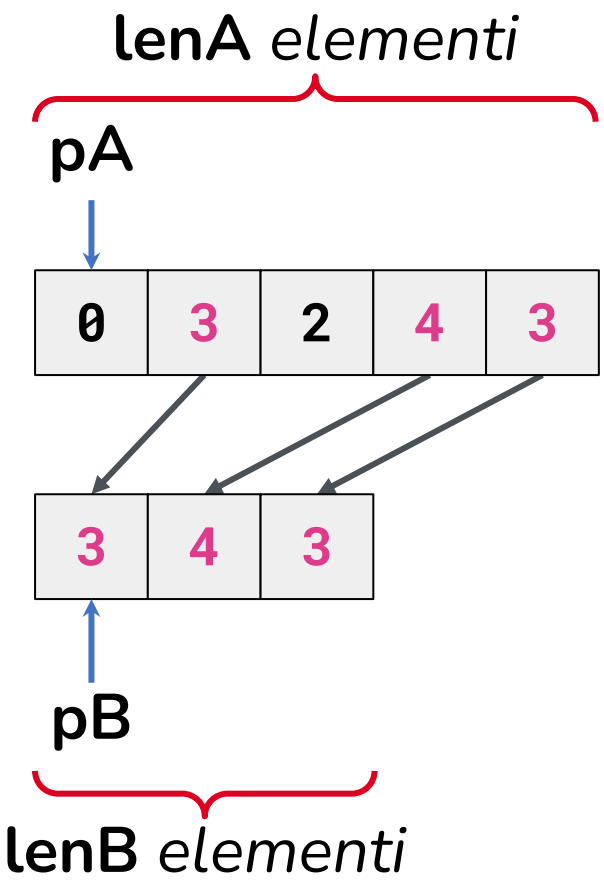
```
size_t lenB = 0;
for (size_t iA=0; iA<lenA; iA++) {
    if (Cond(pA[iA])) {
        // copia pA[iA] in pB
        pB[lenB] = pA[iA];
        lenB++;
    }
}
// alla fine del ciclo lenB è il numero di elementi copiati in pB
```

NOTA: il codice assume che l'array puntato da **pB** abbia sufficiente spazio per accomodare gli elementi copiati. Vediamo in seguito un miglioramento.

Copia selettiva elementi array

Dato un puntatore **pA** ad una sequenza di dati di lunghezza **lenA**, vogliamo copiare in un'altra sequenza **pB** tutti gli elementi che rispettano un criterio *Cond*. Come fare?

Esempio:

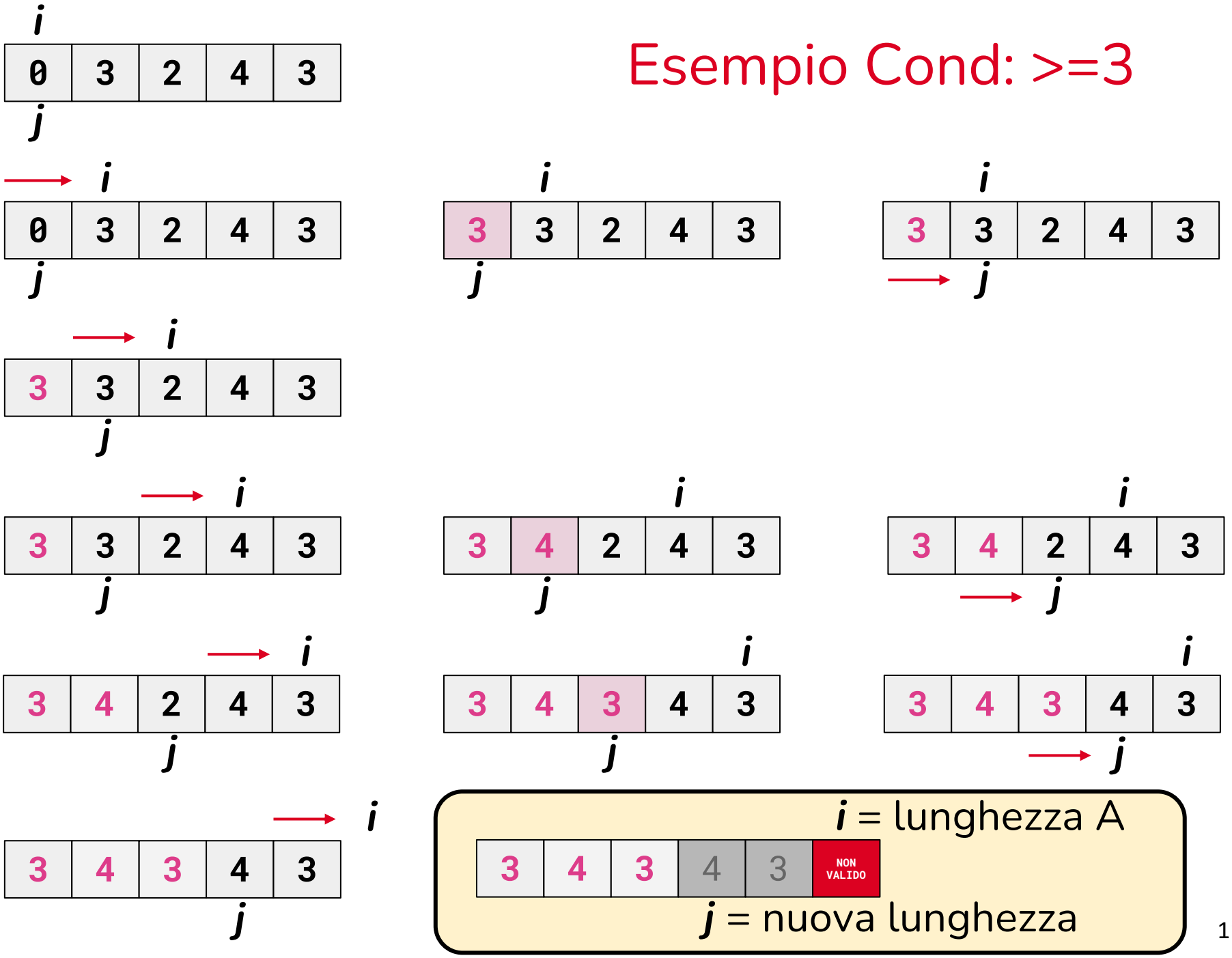


Riduzione sul posto (*in-place removal*)

Dato un puntatore **pA** ad una sequenza di dati di lunghezza **lenA**, vogliamo ridurre la sequenza mantenendo solo gli elementi che rispettano un criterio *Cond*. Come fare?

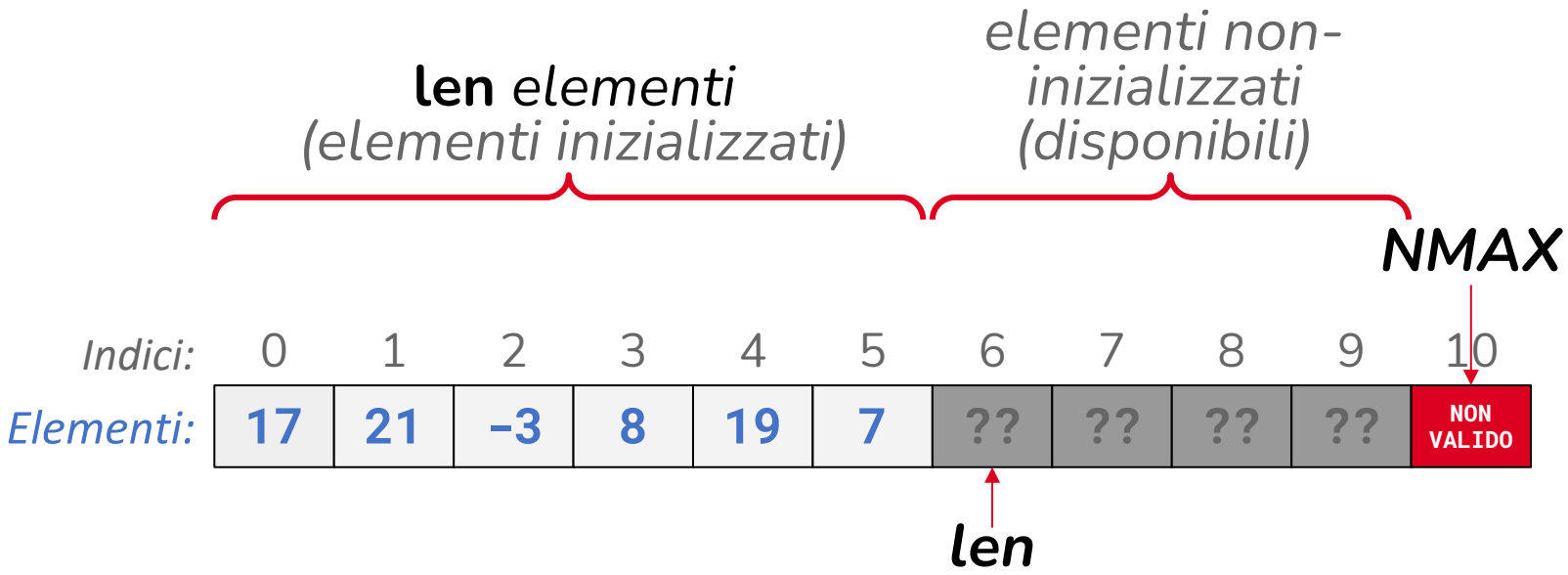
```
size_t j = 0;
for (size_t i=0; i<lenA; i++) {
    if (Cond(pA[i])) {
        // copia pA[i] in pA[j]
        pA[j] = pA[i];
        j++;
    }
}
// alla fine del ciclo j è il numero di elementi rimasti
lenA = j;
```

Esempio Cond: >=3



Buffer pre-allocato

Consideriamo il modello di memoria:



Proprietà:

- Indici validi:

  [0, len)

  (quindi un indice **i** *ad un elemento inizializzato rispetta:* **0 ≤ i < len***)*

- Elementi memorizzabili nel buffer: **0 ≤ len ≤ NMAX**

  (possiamo quindi aumentare o diminuire **len***, rispettando il vincolo che il numero massimo di elementi rimane* **NMAX***).*

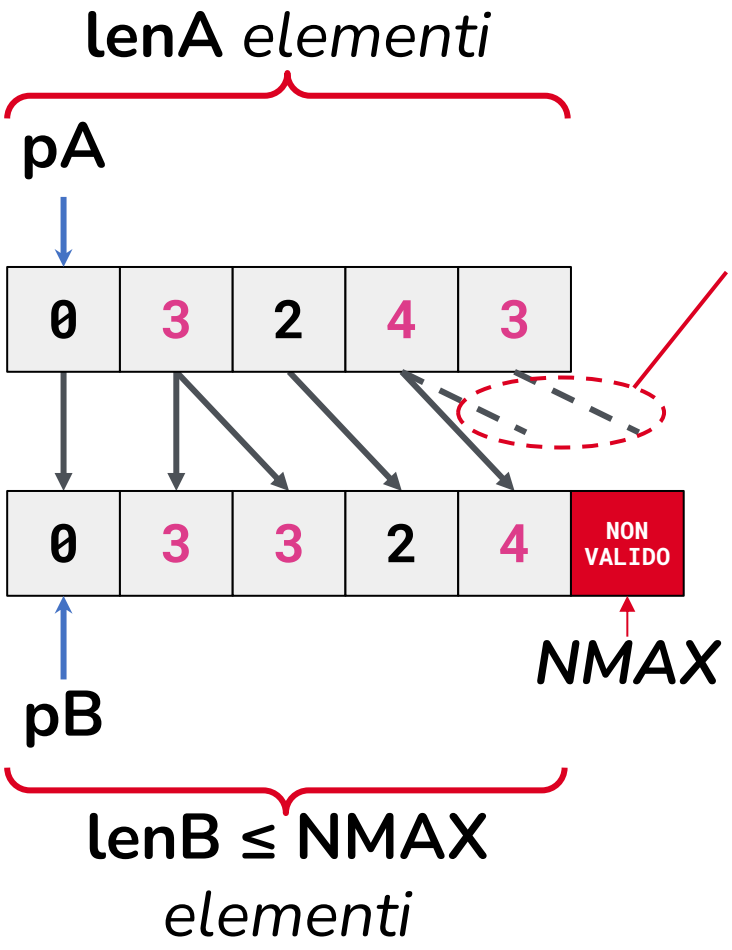
Duplicazione selettiva di elementi

Dato un puntatore **pA** ad un array di dati di lunghezza **lenA**, vogliamo copiare in un altro array **pB** con buffer di dimensione **NMAX** tutti gli elementi di **pA**, copiando due volte gli elementi che rispettano un criterio *Cond*. Come fare?

```
size_t lenB = 0;
for (size_t iA=0; iA<lenA && lenB<NMAX; iA++) {
    pB[lenB] = pA[iA]; // copia pA[iA] in pB
    lenB++;
    if (Cond(pA[iA]) && lenB<NMAX) {
        pB[lenB] = pA[iA]; // copia una seconda volta pA[iA] in pB
        lenB++;
    }
}
// alla fine del ciclo lenB è il numero di elementi copiati in pB
```

Duplicazione selettiva di elementi

Esempio:



Gli elementi che non stanno nel buffer di **pB** non vengono copiati/duplicati.

Modifiche ad array

Aprire il file **modifica_array.c** fornito nel codice iniziale, leggere il contenuto ed infine implementare le funzioni dichiarate, seguendo la specifica.

Trasformazione seguendo regole

Sulla pagina Moodle trovate un esercizio con nome

Lab07-Es1 Regole trasformazione array

Leggere la specifica ed il codice, e completare la funzione **trasforma_array**. Il main e le funzioni per leggere l'input e scrivere l'output sono già fornite.

Aritmetica dei puntatori con le stringhe

Dato che le stringhe sono array di char convenzionalmente terminati dal carattere `'\0'`, sono molto facili da manipolare tramite aritmetica dei puntatori.

Passaggio come argomento:

```
void funzione(const char* pStrInput)
```

Iterazione su tutti i caratteri:

```
while (*pStr != '\0') {
    /* usa *pStr */
    pStr++;
}
```

```
for ( ; *pStr != '\0'; pStr++) {
    /* usa *pStr */
}
```

Accediamo ai caratteri tramite dereferenziamento.

Puntatori a stringhe

Aprire il file **codifica.c**, leggere il codice, e completare la richiesta. Vi viene chiesto di scrivere due funzioni:

- **codifica**: prende in ingresso una stringa e ne altera *in-place* le lettere maiuscole secondo queste regole:
  A→B, B→C, C→D, … Y→Z, Z→A
  Esempio: la stringa “ABC” viene trasformata in “BCD”
- **decodifica**: effettua la trasformazione inversa.

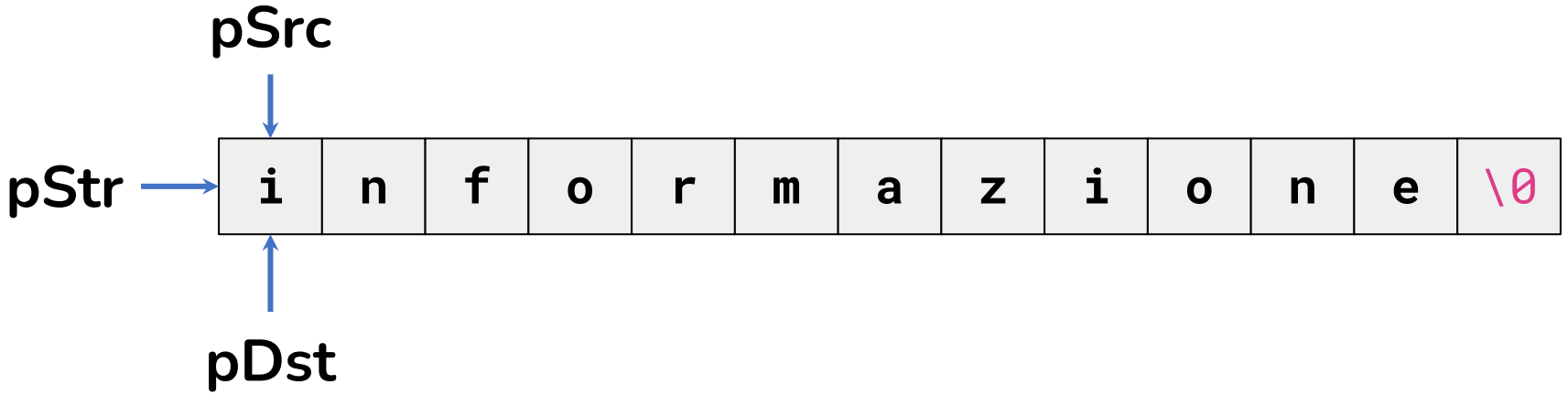
Elimina caratteri [1/4]

Vogliamo scrivere una funzione **elimina_vocali** che prende in ingresso una stringa ed elimina *in-place* tutti i caratteri che sono vocali, “compattando” i rimanenti in una stringa uguale o più piccola.

Esempio: **elimina_vocali**(“informazione”) ottiene “nfrmzn”

Come facciamo?

Elimina caratteri [2/4]

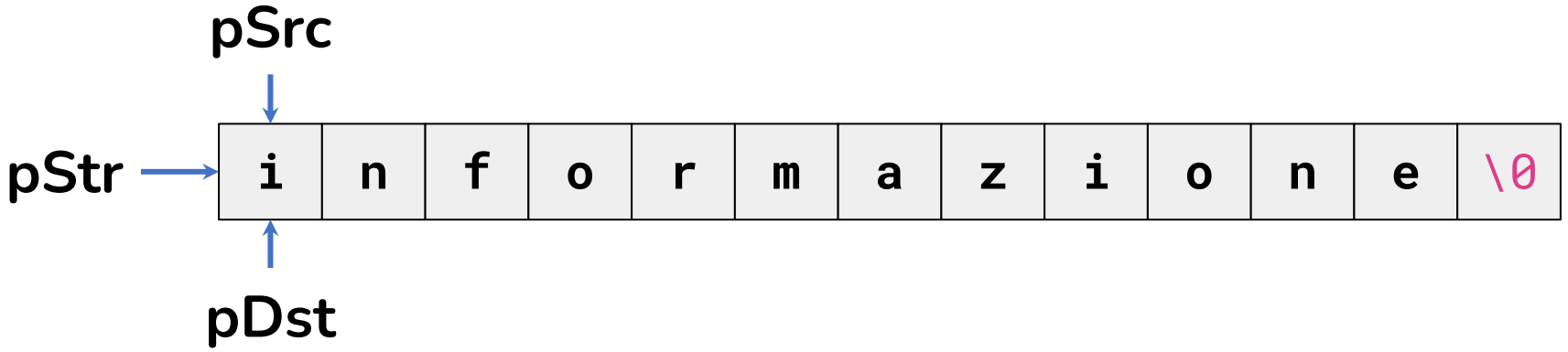


Usiamo due puntatori aggiuntivi **pSrc** e **pDst**:

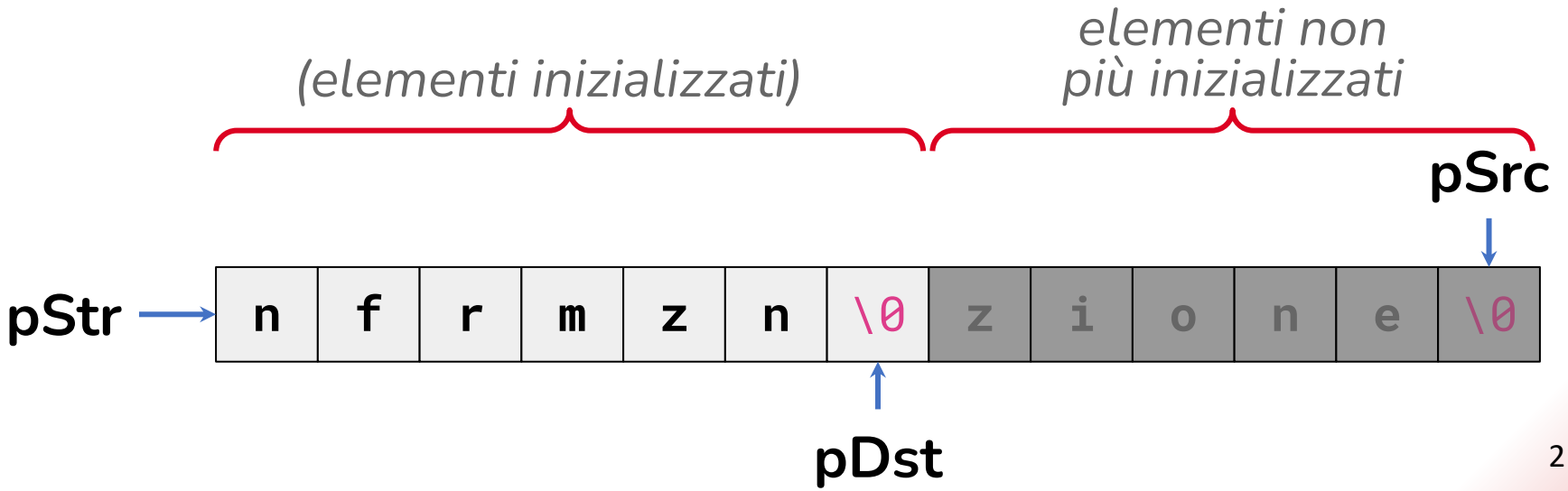
- **pSrc** e **pDst** vengono inizializzati a **pStr**:
- Il primo (**pSrc**) scorre tutta la stringa
- Il secondo (**pDst**) copia ***pSrc** ed avanza soltanto quando il carattere rispetta il criterio di mantenimento.
- Alla fine, scriviamo in **pDst** il terminatore (quindi il terminatore potrebbe cadere in una posizione precedente a quella iniziale).

Elimina caratteri [3/4]

Prima dell'eliminazione delle vocali:



Dopo l'eliminazione delle vocali:



Elimina caratteri [4/4]

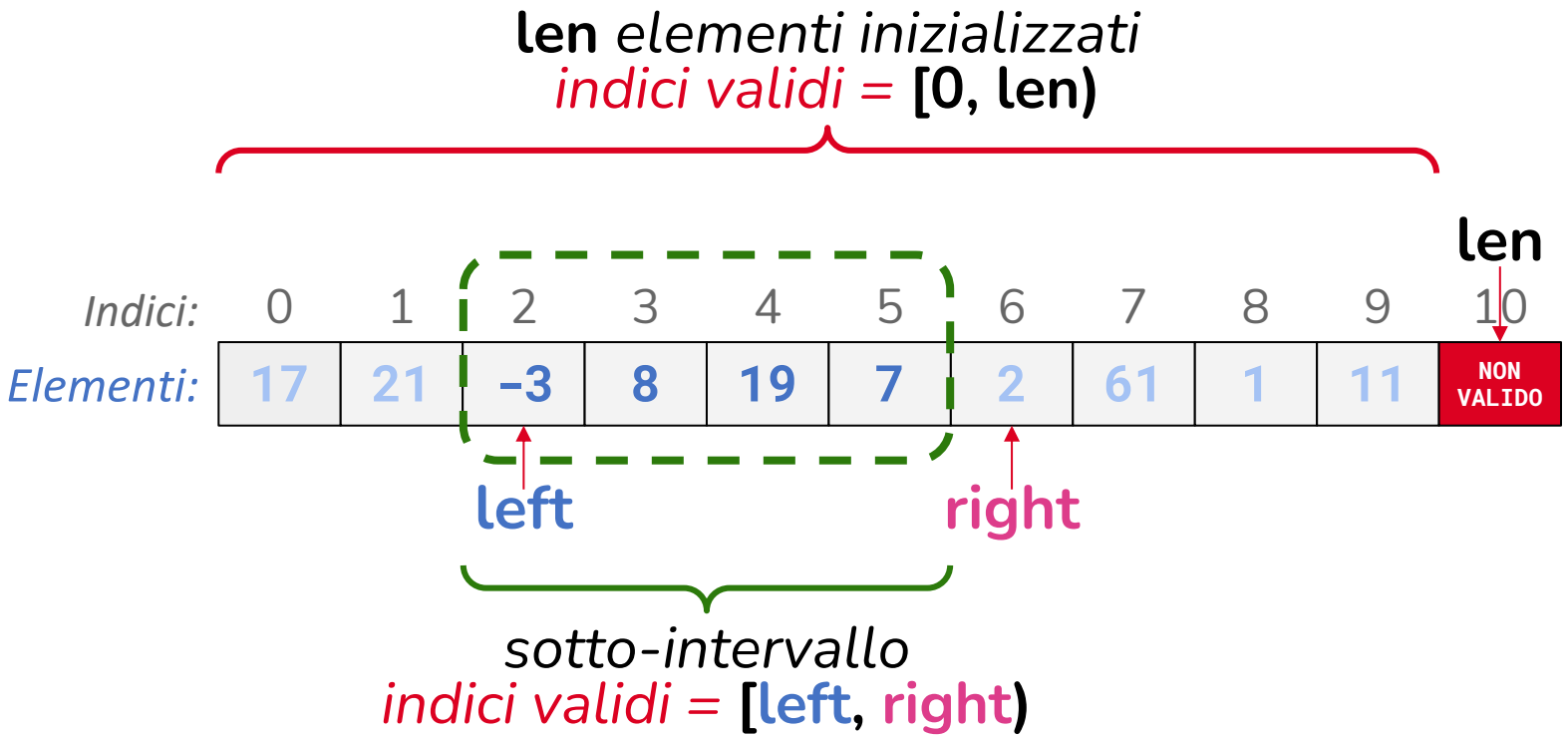
Sulla pagina Moodle trovate un esercizio con nome

Lab07-Es2 Elimina vocali

Leggere la specifica e scrivere il codice.

Intervalli semiaperti di indici

Generalizziamo la nozione di *spazio degli indici validi*:



Caso base: **[left, right) == [0, len)**

Caso generale: **[left, right) ⊆ [0, len)**

cioè vale: **0 ≤ left ≤ right ≤ len**

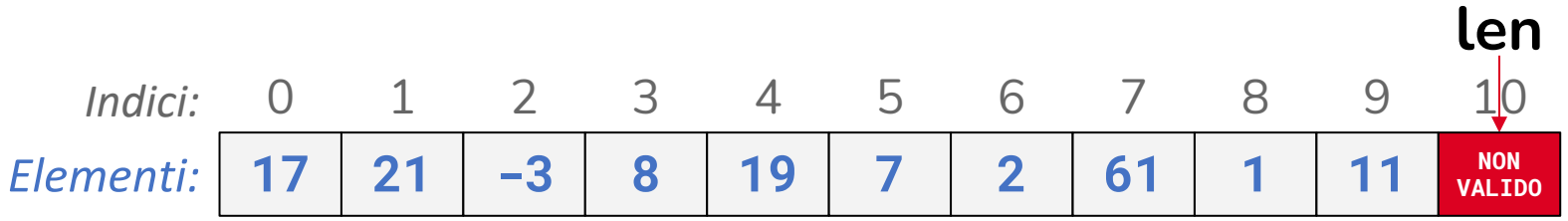
Intervalli semiaperti di indici

Dato un intervallo semiaperto

[0, len)

di indici di un array **arr**[], possiamo dire:

- Il numero di elementi dell'intervallo è: **len**
- L'intervallo è vuoto se: **0==len**
- Un indice **i** appartiene all'intervallo se: **0 ≤ i < len**
- Se l'intervallo non è vuoto, allora:
  - il primo elemento è: **arr[0]**
  - l'ultimo elemento è: **arr[len - 1]**
- Deve sempre valere l'invariante: **assert(len >= 0);**



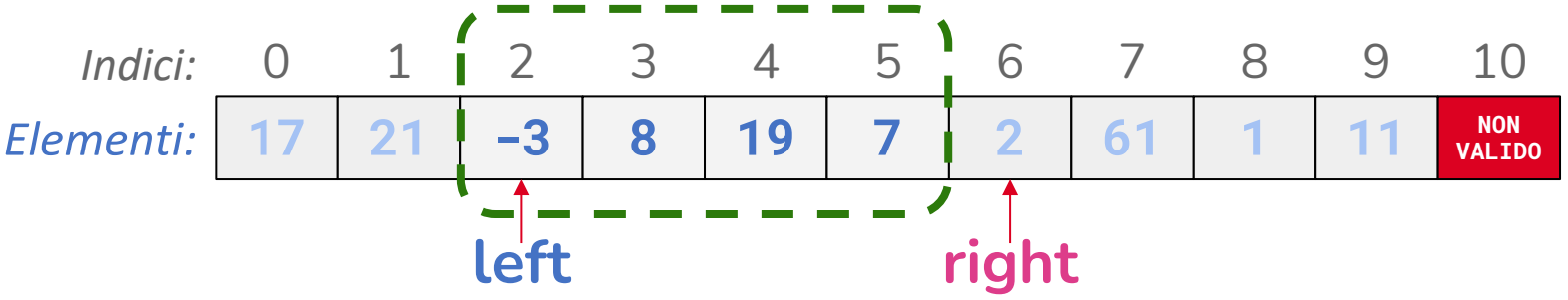
Intervalli semiaperti generali di indici

Dato un intervallo semiaperto

[left, right)

di indici di un array **arr**[], possiamo dire:

- Il numero di elementi dell'intervallo è: **right - left**
- L'intervallo è vuoto se: **left==right**
- Un indice **i** appartiene all'intervallo se: **left ≤ i < right**
- Se l'intervallo non è vuoto, allora:
  - il primo elemento è: **arr[left]**
  - l'ultimo elemento è: **arr[right - 1]**
- Deve sempre valere l'invariante: **assert(right >= left);**



Intervalli semiaperti generali di puntatori

Dato un intervallo semiaperto

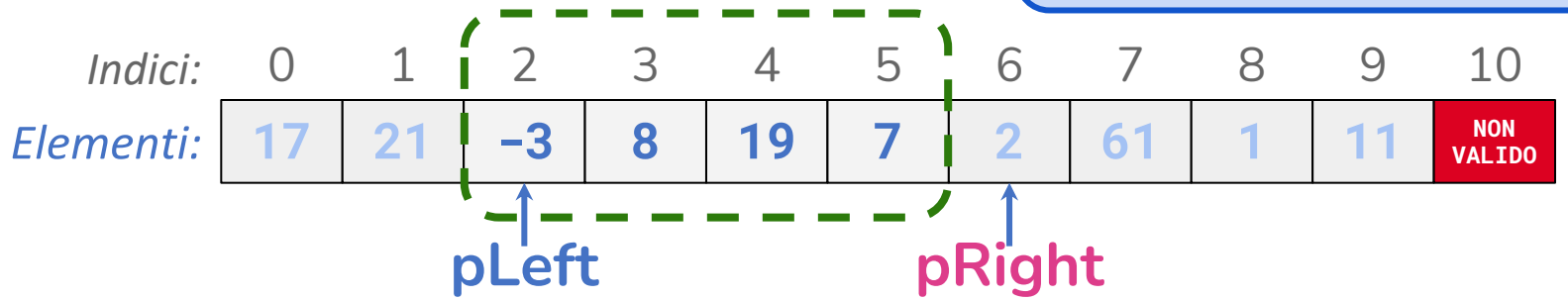
[pLeft, pRight)

di **puntatori** agli elementi in un array **arr**[], possiamo dire:

- Il numero di elementi dell'intervallo è: **pRight - pLeft**
- L'intervallo è vuoto se: **pLeft==pRight**
- Un puntatore **p** appartiene all'interv.: **pLeft ≤ p < pRight**
- Se l'intervallo non è vuoto, allora:
  - il primo elemento è: ***pLeft**
  - l'ultimo elemento è: ***(pRight - 1)**
- Deve sempre valere l'invariante:

assert(pRight >= pLeft);

Non tratteremo il caso di intervalli di puntatori.



Usi degli intervalli

Esempi di usi degli intervalli

- Identificare sotto-intervalli di array con certe proprietà.
- Cercare una sottosequenza dentro un array.
- Ricerca binaria (dicotomica)

In generale: molti problemi diversi su indici, puntatori, intervalli operano con logiche comuni → cercate di essere consistenti nell'applicare e riusare un numero ridotto di schemi ricorrenti e consolidati.

Funzioni su stringhe con intervalli!

Leggere il codice del programma **my_string.c** ed implementare le funzioni commentate nella dichiarazione dei prototipi.

 Senza usare le funzioni di `<string.h>`!

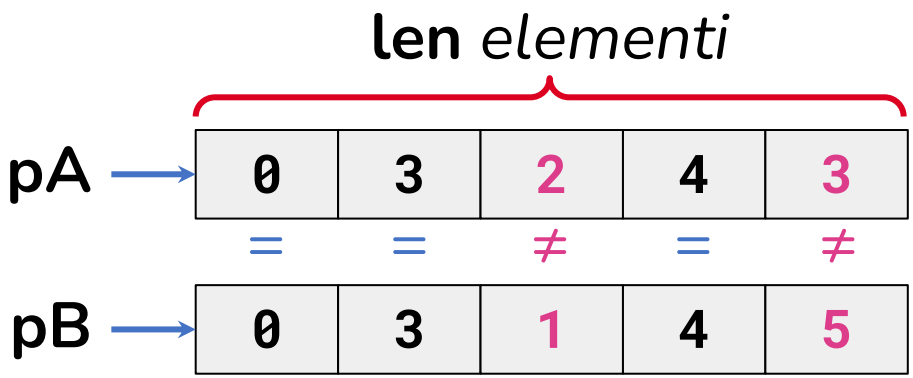
Sottosequenze con errori

ESERCIZIO EXTRA

Leggere l'esercizio su Moodle.

Lab07-Es3 Sottosequenze con errori

Date due sequenze **A** e **B** della stessa lunghezza **len**, diciamo che **A** e **B** sono in **match con al più n errori** se le coppie di elementi nelle medesime posizioni sono uguali, con al più **n** coppie di elementi differenti.



match con 2 errori.