UNIVERSITÀ DI TORINO

Laboratorio di Programmazione I

Lezione n. 9: Ricorsione

Alessandro Mazzei

Slides: Elvio Amparore

Outline

- Ricorsione
- Ricorsione aritmetica/algebrica
- Quantificazione universale/esistenziale ricorsiva
- aritmeticaR.c
- Ricorsione su stringhe
- base_stringheR.c e stringheR.c
- Lab09-Es1 Delezione basi
- Ricorsione su array
- base_arrayR.c e arrayR.c
- Lab09-Es2 Sottoinsieme ricorsivo
- Lab09-Es3 Slalom sciistico (extra)

Outline

- Ricorsione
- Ricorsione aritmetica/algebrica
- Quantificazione universale/esistenziale ricorsiva
- aritmeticaR.c
- Ricorsione su stringhe
- base_stringheR.c e stringheR.c
- Lab09-Es1 Delezione basi
- Ricorsione su array
- base_arrayR.c e arrayR.c
- Lab09-Es2 Sottoinsieme ricorsivo
- Lab09-Es3 Slalom sciistico (extra)

Ricorsione

Funzione ricorsiva: fnR(..., int n)

- **Caso base**: condizione che fa terminare la ricorsione
- **Passo induttivo**: operazione che svolge un passo per l'elemento **n**-esimo, richiamando la stessa funzione su un problema più piccolo.
  Ha tipicamente due parti:
  - richiama **fnR**() con un input diverso (più vicino al caso base)
  - combina i risultati della computazione locale al passo **n** con il risultato della chiamata ricorsiva.

- **Wrapper** (involucro): se l'inizio della ricorsione richiede un'inizializzazione complessa, si definisce una funzione apposita [*necessaria in molti casi*]

Tipologie di ricorsione che consideriamo

- **Ricorsione diretta**: la funzione **fnR** richiama se stessa.
  Esistono problemi che però richiedono forme di ricorsione in cui si alternano funzioni diverse (es: **fnR** chiama **fnR2** che chiama **fnR**…) dette ricorsioni mutue.
  ➡️ Considereremo solo ricorsioni dirette.

- **Ricorsione primitiva**: ogni volta che la funzione **fnR** richiama se stessa con un nuovo input, ci si avvicina progressivamente al caso base.
  Questo garantisce in modo semplice la terminazione.
  ➡️ Considereremo solo ricorsioni primitive.

- **Ricorsione lineare**: la funzione **fnR** esegue al più una sola chiamata ricorsiva al suo interno, e quindi la sequenza delle chiamate forma una catena lineare.
  ➡️ Considereremo principalmente ricorsioni lineari, ma vedremo alcuni esempi non-lineari (es: Fibonacci).

Tipologie di ricorsione che consideriamo

In molti casi comuni la determinazione del caso base dipende da un valore/indice, detto indice della ricorsione.

L'indice della ricorsione è il valore che cambia ad ogni chiamata ricorsiva. Permette di identificare il caso base (che corrisponde ad un valore limite).

- Ricorsione **decrescente** (o **covariante, right-to-left**): L'indice della ricorsione scende ad ogni passo della ricorsione. Il caso base è tipicamente lo **0**.
- Ricorsione **crescente** (o **controvariante, left-to-right**): L'indice della ricorsione cresce ad ogni passo della ricorsione. Il caso base è tipicamente un limite superiore $\mathbf{n}_{max}$ (es.: la dimensione di un array).

Ricorsione aritmetica

Ricorsione aritmetica sui naturali

- La ricorsione è aritmetica/algebrica quando l'indice di ricorsione è un numero naturale (talvolta un intero).
- La ricorsione aritmetica è tipicamente **covariante**:
  - si parte da un valore iniziale **n** == $\mathbf{n}_{max}$
  - ogni passo della ricorsione decrementa **n**
  - il caso base è (in genere) lo **0**.

Ricorsione **covariante** algebrica

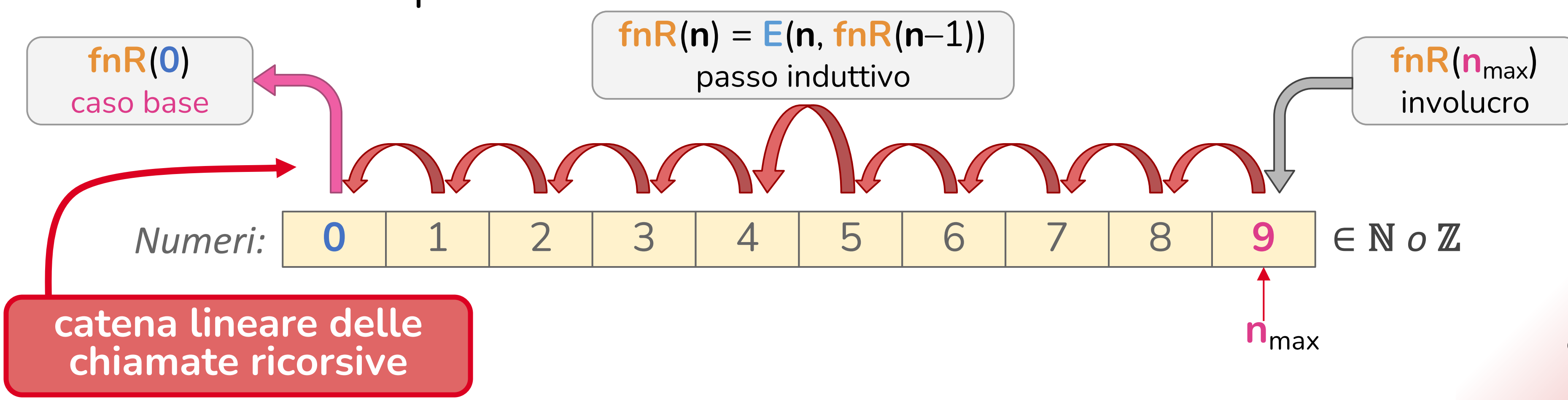
Sia $\mathbf{n}_{max} \in \mathbb{N}$ un numero naturale.

Ricorsione covariante: indice **n** <u>decrescente</u>

- Caso base: **n** == 0 ⇒ valore base
- Passo induttivo: **n** > 0 ⇒ Ricorsione con **n** – 1
- Involucro: iniziamo con **n** == $\mathbf{n}_{max}$

$$\mathrm{fnR}(\mathbf{n}) = \begin{cases} \mathrm{fnR}(0) = \mathbf{val}_{base} & \mathbf{n}==0 \\ \mathrm{fnR}(\mathbf{n}) = \mathrm{E}(\mathbf{n}, \mathrm{fnR}(\mathbf{n}-1)) & \mathbf{n} > 0 \end{cases}$$

L'operazione E combina il valore calcolato al passo **n** con il risultato della ricorsione per **n**–1.



Ricorsione **controvariante** algebrica

La ricorsione aritmetica che considereremo sarà quasi solo covariante

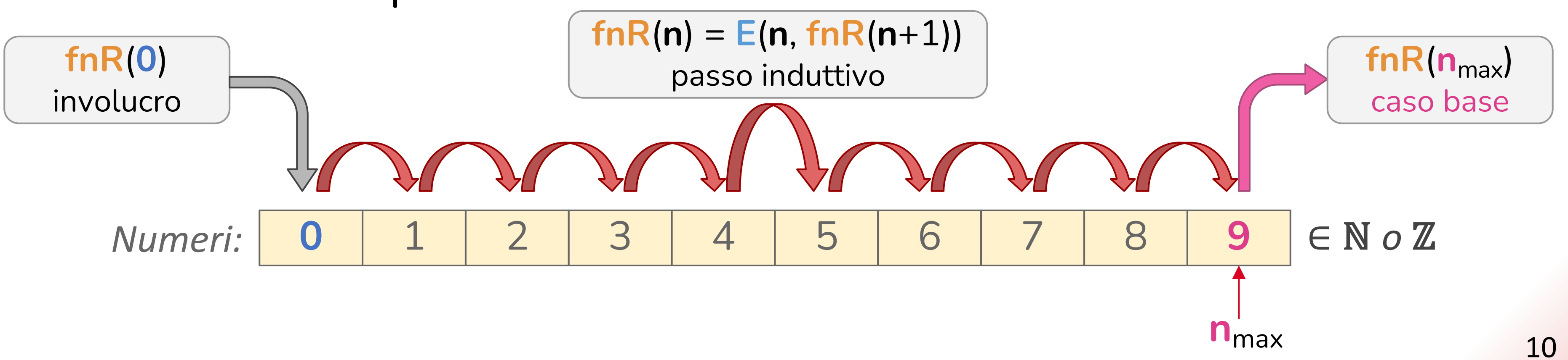
Sia $\mathbf{n}_{max} \in \mathbb{N}$ un numero naturale.

Ricorsione controvariante: indice **n** crescente

- Caso base: $\mathbf{n} == \mathbf{n}_{max}$ ⇒ valore base
- Passo induttivo: $\mathbf{n} < \mathbf{n}_{max}$ ⇒ Ricorsione con $\mathbf{n} + 1$
- Involucro: iniziamo con $\mathbf{n} == 0$

$$\mathrm{fnR}(\mathbf{n}) = \begin{cases} \mathrm{fnR}(\mathbf{n}_{max}) = \mathbf{val}_{base} & \mathbf{n}==\mathbf{n}_{max} \\ \mathrm{fnR}(\mathbf{n}) = \mathrm{E}(\mathbf{n}, \mathrm{fnR}(\mathbf{n}+1)) & \mathbf{n} < \mathbf{n}_{max} \end{cases}$$

L'operazione **E** combina il valore calcolato al passo **n** con il risultato della ricorsione per **n**+1.



Somma pari

Esempio: calcoliamo la sommatoria dei numeri pari nell'intervallo da 0 ad **n**.

$$\mathrm{fnR}(\mathbf{n}) = \begin{cases} \mathrm{fnR}(0) = \mathbf{0} & \mathbf{n}{=}{=}0 \\ \mathrm{fnR}(\mathbf{n}) = \mathrm{fnR}(\mathbf{n}-1) + \text{n se pari} & \mathbf{n} > 0 \end{cases}$$

```
int sommaPariR(int n) {
    if (n==0) { // caso base
        return 0;
    }
    else {
        if (n%2==0) // pari
            return n + sommaPariR(n-1);
        else // dispari
            return sommaPariR(n-1);
    }
}
```

Numero primo

Determiniamo se **m** è primo. Usiamo **n** come indice decrescente di ricorsione

$$\textbf{primoR}(\mathbf{m}, \mathbf{n}) = \begin{cases} \textbf{primoR}(\mathbf{m}, 1) = \textbf{true} & \mathbf{n}==1 \\ \textbf{primoR}(\mathbf{m}, \mathbf{n}) = \text{false se n divide m} & \\ \qquad\qquad \textbf{primoR}(\mathbf{m}, \mathbf{n}-1) \text{ altrimenti} & \mathbf{n} > 0 \end{cases}$$

Caso base: in questo caso è 1.

Wrapper: **primo**(**m**) = **primoR**(**m**, **m**-1)

NOTA: **m** non cambia mai durante la ricorsione (è un parametro invariante della ricorsione)

Numero primo

$$\text{primoR}(\mathbf{m}, \mathbf{n}) = \begin{cases} \text{primoR}(\mathbf{m}, 1) = \textbf{true} & \mathbf{n}{==}1 \\ \text{primoR}(\mathbf{m}, \mathbf{n}) = \text{false se n divide m} & \\ \qquad\qquad\qquad \text{primoR}(\mathbf{m}, \mathbf{n}-1) \text{ altrimenti} & \mathbf{n} > 0 \end{cases}$$

```
bool primoR(int m, int n) {
    if (n==1) // caso base
        return true;
    else // passo induttivo
        return (m%n != 0) && primoR(m, n-1);
}

bool primo(int m) { // involucro
    assert(m > 1);
    return primoR(m, m-1);
}
```

Quantificazione universale ricorsiva

Sia ***Cond*** un criterio booleano. Diamo una formulazione ricorsiva che risponde a questa domanda:

Tutti i numeri della sequenza
soddisfano una condizione ***Cond***?

Ricorsione covariante:

$$\mathbf{fnR}(\mathbf{n}) = \begin{cases} \mathbf{fnR}(0) = \mathbf{true} & \mathbf{n}{=}{=}0 \\ \mathbf{fnR}(\mathbf{n}) = \mathit{Cond}(\mathbf{n}) \boxed{\&\&} \mathbf{fnR}(\mathbf{n}-1) & \mathbf{n} > 0 \end{cases}$$

Ricorsione controvariante:

$$\mathbf{fnR}(\mathbf{n}) = \begin{cases} \mathbf{fnR}(\mathbf{n}_{max}) = \mathbf{true} & \mathbf{n}{=}{=}\mathbf{n}_{max} \\ \mathbf{fnR}(\mathbf{n}) = \mathit{Cond}(\mathbf{n}) \boxed{\&\&} \mathbf{fnR}(\mathbf{n}+1) & \mathbf{n} < \mathbf{n}_{max} \end{cases}$$

cortocircuitazione degli operatori logici

Quantificazione esistenziale ricorsiva

Sia ***Cond*** un criterio booleano. Diamo una formulazione ricorsiva che risponde a questa domanda:

Esiste almeno un numero della sequenza
che soddisfa una condizione ***Cond***?

Ricorsione covariante:

$$\mathrm{fnR}(\mathbf{n}) = \begin{cases} \mathrm{fnR}(0) = \textbf{false} & \mathbf{n}==0 \\ \mathrm{fnR}(\mathbf{n}) = \mathit{Cond}(\mathbf{n}) \,||\, \mathrm{fnR}(\mathbf{n}-1) & \mathbf{n} > 0 \end{cases}$$

Ricorsione controvariante:

$$\mathrm{fnR}(\mathbf{n}) = \begin{cases} \mathrm{fnR}(\mathbf{n}_{max}) = \textbf{false} & \mathbf{n}==\mathbf{n}_{max} \\ \mathrm{fnR}(\mathbf{n}) = \mathit{Cond}(\mathbf{n}) \,||\, \mathrm{fnR}(\mathbf{n}+1) & \mathbf{n} < \mathbf{n}_{max} \end{cases}$$

Anche in questo caso sfruttiamo la cortocircuitazione degli operatori logici.

La funzione involucro (wrapper, guscio)

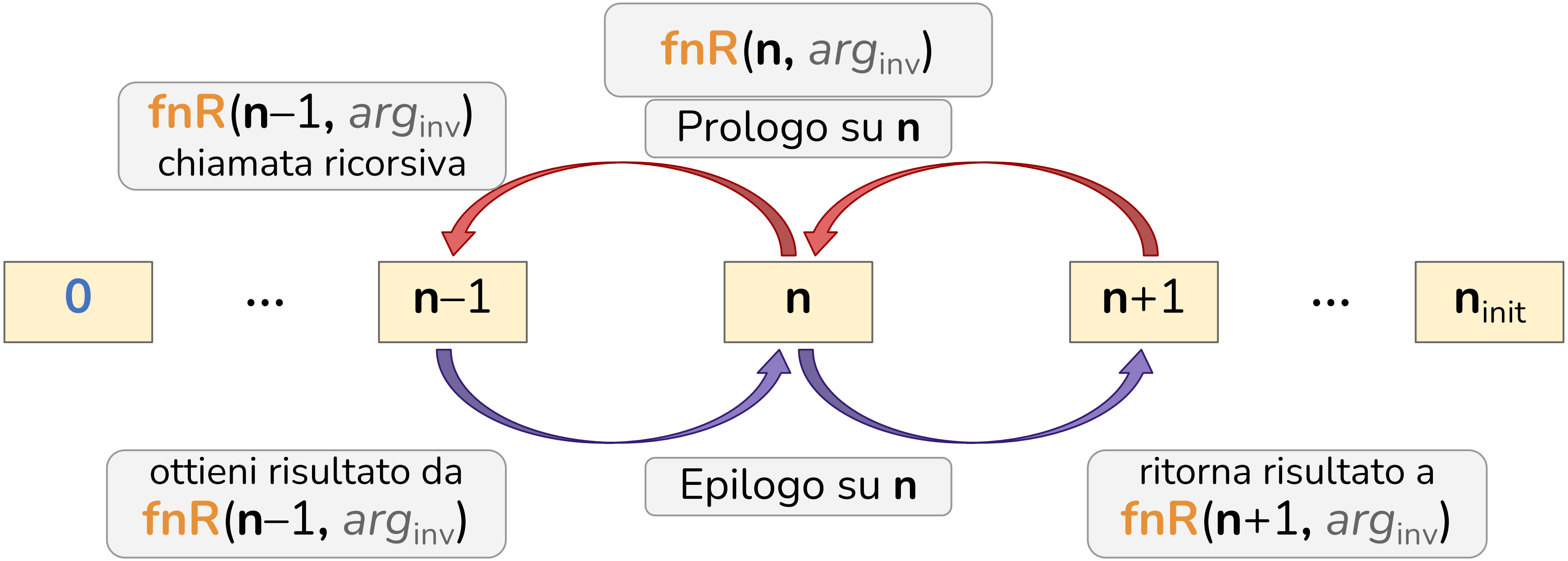
Funzione involucro: funzione ausiliaria che semplifica l'interfaccia (cioè gli argomenti) che servono per avviare la prima chiamata ricorsiva.

- Inizializza gli indici di ricorsione con il valore iniziale;
- Passa inalterati i parametri invarianti;
- Controlla la validità degli argomenti passati;
- Gestisce eventuali operazioni preliminari sugli argomenti.

Lo scopo è sistemare in un unico punto la preparazione della chiamata ricorsiva, rendendo il codice più ordinato e leggibile.

- **Esempio**: l'involucro `primo(m)` chiama `primoR(m,n)` inizializzando l'indice di ricorsione `n` con `m-1`.
- Le funzioni ricorsive su array tipicamente necessitano di un involucro.

Passo induttivo della ricorsione lineare
(caso covariante)



arg_{inv} = parametri invarianti

L'epilogo dispone del valore ritornato da **fnR**(**n**–1, arg_{inv}).

Quando l'epilogo esegue, la ricorsione è già arrivata fino al caso base, e sta "tornando indietro".

Esercizi sulla ricorsione aritmetica

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

SUGGERIMENTO: identificate prima il caso base e il passo induttivo. Per la prima funzione (**mcd_euclideR**) le formule per il caso base ed il passo induttivo sono già fornite come esempio.

Outline

- Ricorsione
- Ricorsione aritmetica/algebrica
- Quantificazione universale/esistenziale ricorsiva
- aritmeticaR.c
- Ricorsione su stringhe
- base_stringheR.c e stringheR.c
- Lab09-Es1 Delezione basi
- Ricorsione su array
- base_arrayR.c e arrayR.c
- Lab09-Es2 Sottoinsieme ricorsivo
- Lab09-Es3 Slalom sciistico (extra)

Ricorsione su stringhe

Ricorsione **controvariante** su stringhe

Le funzioni che adoperano l'aritmetica dei puntatori sulle stringhe si prestano in modo naturale ad essere trattate con la ricorsione controvariante (crescente, SxToDx).

Consideriamo una funzione ricorsiva con argomento:

```
fnR(const char* pStr)
```

Abbiamo i seguenti casi:

- **Caso base**:
  - `*pStr=='\0'` (raggiunto il terminatore)
- **Passo induttivo**:
  - leggiamo il valore: `*pStr` (dereferenziamento)
  - ricorsione su: `pStr+1` (avanzamento puntatore)
- Inizio: **pStr** punta al primo carattere della stringa.

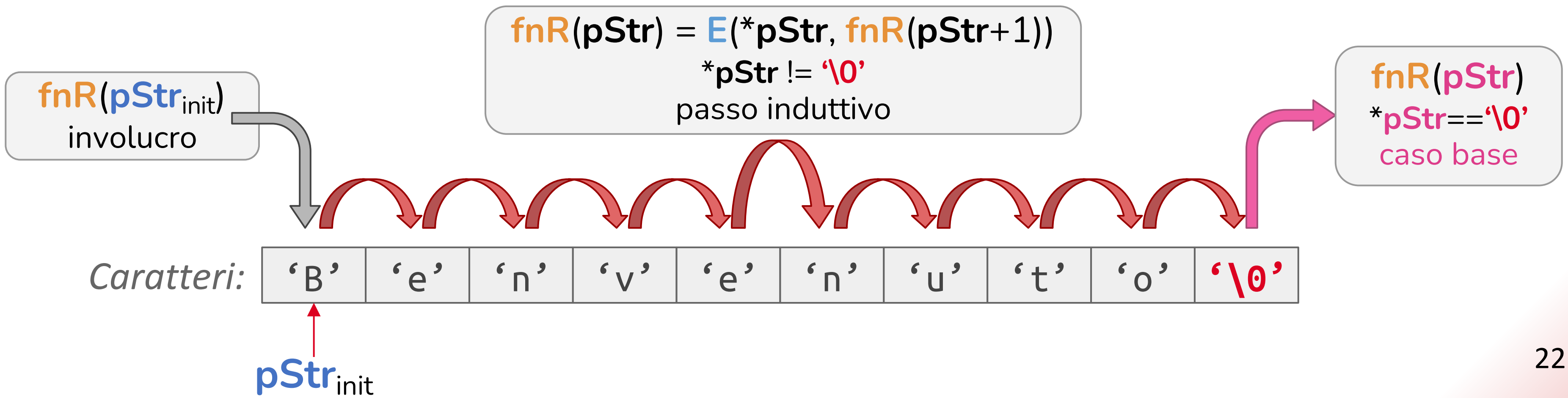
Ricorsione **controvariante** su stringhe

Sia **pStr**$_{init}$ un puntatore ad una stringa.

Ricorsione controvariante: puntatore **pStr** crescente

- Caso base: ***pStr == '\0'** ⇒ valore base
- Passo induttivo: ***pStr** != **'\0'** ⇒ Ricorsione con **pStr** + 1
- Involucro: non serve (iniziamo direttamente con **pStr**$_{init}$)

La ricorsione su stringhe che considereremo sarà quasi sempre controvariante



Esercizi sulla ricorsione con le stringhe

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

- per la stampa delle stringhe (normale/capovolta), ragionare sull'ordine delle operazioni di stampa e di chiamata ricorsiva
- per la funzione di test di uguaglianza, riflettere sui due casi:
  - confrontare carattere per carattere delle due stringhe
  - qual è il caso base?

Esercizi sulla ricorsione con le stringhe

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

Delezione basi

In genetica una **delezione di basi** è la perdita di una o più basi azotate (A,T,C,G) in una sequenza di DNA.

ATTACGGTCATAGGCGATGGAC

ATTACG–AGGCGATGGAC [delezione di **GTCAT**]

Sulla pagina Moodle trovate un esercizio con nome

Lab09-Es1 Delezione basi

Completate il programma.

NOTA: tutte le funzioni richieste devono essere ricorsive. Non potete scrivere cicli for/while.

Outline

- Ricorsione
- Ricorsione aritmetica/algebrica
- Quantificazione universale/esistenziale ricorsiva
- aritmeticaR.c
- Ricorsione su stringhe
- base_stringheR.c e stringheR.c
- Lab09-Es1 Delezione basi
- <u>Ricorsione su array</u>
- <u>base_arrayR.c e arrayR.c</u>
- <u>Lab09-Es2 Sottoinsieme ricorsivo</u>
- Lab09-Es3 Slalom sciistico (extra)

Ricorsione su array

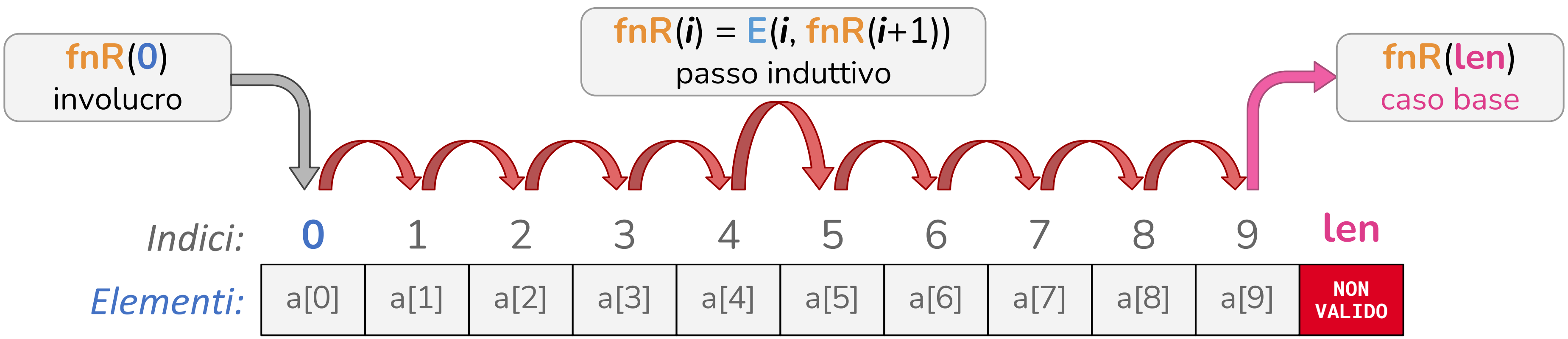
Ricorsione **controvariante** con indici in intervalli semiaperti

Consideriamo un intervallo semiaperto di indici

$$i \in [0, \text{len})$$

Ricorsione controvariante: indice ***i*** crescente

- Caso base: ***i*** ≥ **len** ⇒ valore base o su array vuoto
- Passo induttivo: uso **a**[***i***] ⇒ Ricorsione con ***i*** + 1
- Involucro: iniziamo con ***i*** == **0**



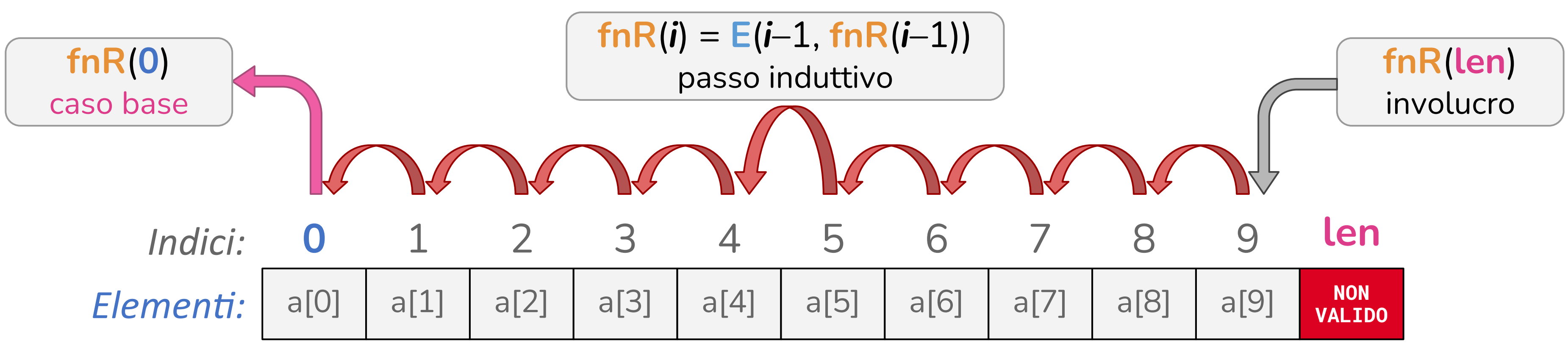
Ricorsione **covariante**
con indici in intervalli semiaperti

Consideriamo un intervallo semiaperto di indici

$$\boldsymbol{i} - 1 \in [0, \text{len})$$

Ricorsione covariante: indice ***i*** <u>decrescente</u>

- Caso base: ***i*** == **0** ⇒ valore base o su array vuoto
- Passo induttivo: uso **a**[***i*** – 1] ⇒ Ricorsione con ***i*** – 1
- Involucro: iniziamo con ***i*** == **len**



Modelli di ricorsione su array con indici in intervalli semiaperti

Ricorsione **controvariante**: indice *i* crescente

```
retType fn(const size_t aLen, const int a[]) { // involucro
        return fnR(aLen, a, 0);
}
retType fnR(const size_t aLen, const int a[], const size_t i)
        if (i == aLen)
                return valbase; // caso base
        else
                return E(a[i], fnR(aLen, a, i+1));
```

Ricorsione **covariante**: indice *i* decrescente

```
retType fn(const size_t aLen, const int a[]) { // involucro
        return fnR(aLen, a, aLen);
}
retType fnR(const size_t aLen, const int a[], const ssize_t i)
        if (i < 0)
                return valbase; // caso base
        else
                return E(a[i-1], fnR(aLen, a, i-1));
```

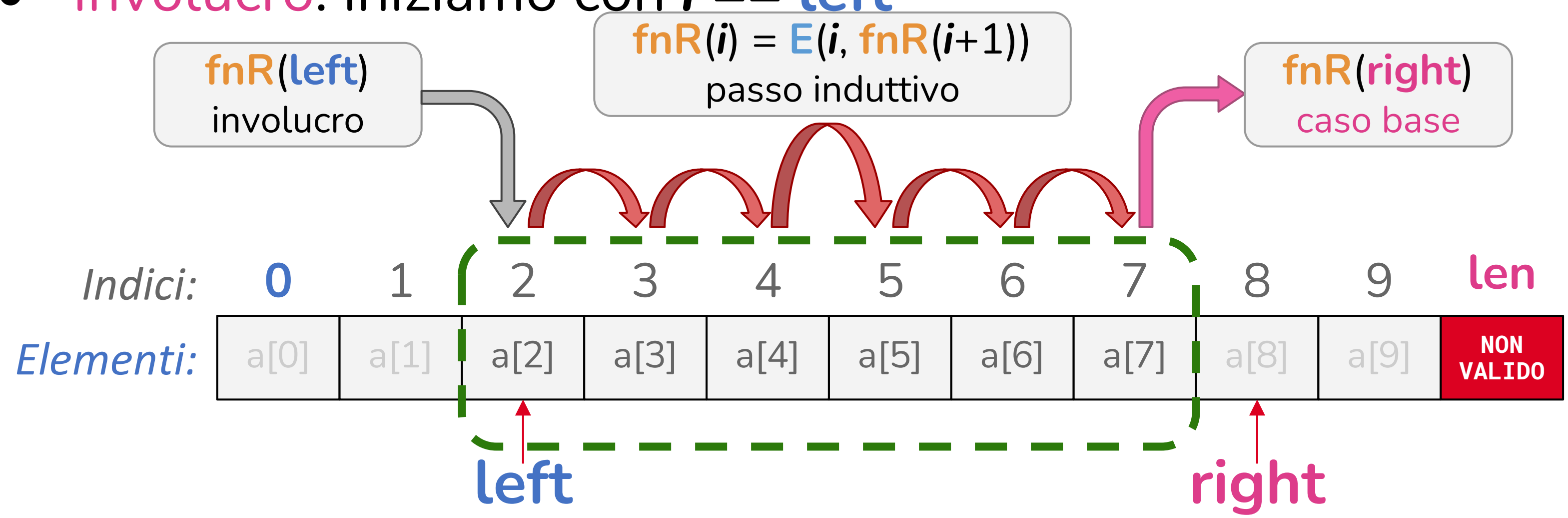
Ricorsione **controvariante** con indici in intervalli semiaperti **generali**

Consideriamo un intervallo semiaperto di indici

i ∈ **[left, right)** con: **0** ≤ **left** ≤ **right** ≤ **len**

Ricorsione controvariante: indice ***i*** crescente

- Caso base: ***i*** ≥ **right** ⇒ valore base o su array vuoto
- Passo induttivo: uso **a**[***i***] ⇒ Ricorsione con ***i*** + 1
- Involucro: iniziamo con ***i*** == **left**



controvariante = da sinistra a destra, o *left-to-right*.

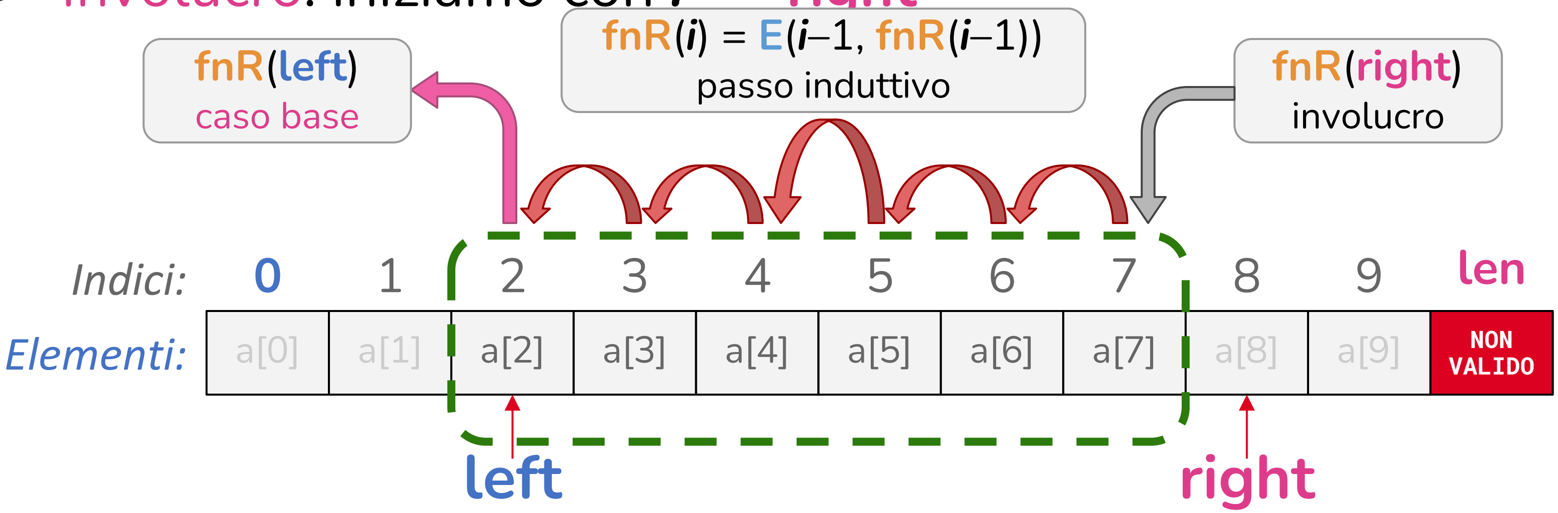
Ricorsione **covariante** con indici in intervalli semiaperti **generali**

Consideriamo un intervallo semiaperto di indici

i – 1 ∈ **[left, right)** con: **0** ≤ **left** ≤ **right** ≤ **len**

Ricorsione covariante: indice ***i*** <u>decrescente</u>

- Caso base: ***i*** == **left** ⇒ valore base o su array vuoto
- Passo induttivo: uso **a**[***i*** – 1] ⇒ Ricorsione con ***i*** – 1
- Involucro: iniziamo con ***i*** == **right**



covariante = da destra a sinistra, o *right-to-left*.

Esercizi di ricorsione su array

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

In seguito implementare gli esercizi nel file **arrayR.c** fornito nel codice iniziale.

Sottoinsiemi di array

Sulla pagina Moodle trovate un esercizio con nome

Lab09-Es2 Sottoinsieme ricorsivo

Completate il programma.

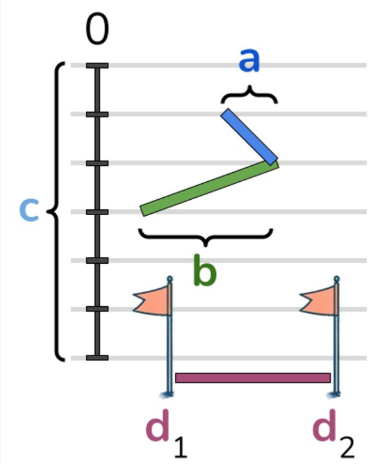
Outline

- Ricorsione
- Ricorsione aritmetica/algebrica
- Quantificazione universale/esistenziale ricorsiva
- aritmeticaR.c
- Ricorsione su stringhe
- base_stringheR.c e stringheR.c
- Lab09-Es1 Delezione basi
- Ricorsione su array
- base_arrayR.c e arrayR.c
- Lab09-Es2 Sottoinsieme ricorsivo
- Lab09-Es3 Slalom sciistico (extra)

Slalom sciistico

ESERCIZIO EXTRA

Lab09-Es3 Slalom sciistico



Uno sciatore, affronta una discesa di lunghezza c. Per ogni unità di c, può decidere se scendere a destra (spostandosi orizzontalmente di a metri) o a sinistra (spostandosi di b metri).

Partendo dalla posizione orizzontale 0, determinare le combinazioni di x discese a destra ed y a sinistra per raggiungere una posizione finale d compresa tra d_1 e d_2, che rappresentano i limiti delle porte del traguardo.

Formulare il problema con l'**algoritmo di Euclide esteso** per risolvere:

A. $a \cdot x + b \cdot y = d$
B. $x + y = c$
C. $x \geq 0,\ \ y \geq 0,\ \ d \in [d_1, d_2)$

La [A.] può essere risolta riconducendola all'identità di Bézout. Usare le slide aggiuntive su Euclide esteso.