

# **Corso di Laurea in INFORMATICA**

**a.a. 2011-2012**

## **Algoritmi e Strutture Dati**

### **MODULO N. 5**

#### **Strutture lineari di dati: Pile e Code**

**Pile e Code: specifiche e realizzazioni attraverso rappresentazioni sequenziali e collegate. Pile e procedure ricorsive. Code e Pile come strutture ausiliarie.**

Questi lucidi sono stati preparati da per uso didattico. Essi contengono materiale originale di proprietà dell'Università degli Studi di Bari e/o figure di proprietà di altri autori, società e organizzazioni di cui e' riportato il riferimento. Tutto o parte del materiale può essere fotocopiato per uso personale o didattico ma non può essere distribuito per uso commerciale. Qualunque altro uso richiede una specifica autorizzazione da parte dell'Università degli Studi di Bari e degli altri autori coinvolti.



# PILE

UNA **PILA** È UNA SEQUENZA DI ELEMENTI DI UN CERTO TIPO IN CUI È POSSIBILE AGGIUNGERE O TOGLIERE ELEMENTI SOLO DA UN ESTREMO DELLA SEQUENZA (LA “**TESTA**”).

PUÒ ESSERE VISTA COME UN CASO SPECIALE DI LISTA IN CUI L'ULTIMO ELEMENTO INSERITO È IL PRIMO AD ESSERE RIMOSSO (**LIFO**) E NON È POSSIBILE ACCEDERE AD ALCUN ELEMENTO CHE NON SIA QUELLO IN TESTA.

Possibili utilizzi:

- ☐ Nei linguaggi con procedure: gestione dei record di attivazione

- ☐ Nei linguaggi stack-oriented:

Le operazioni elementari prendono uno-due operandi dallo stack e inseriscono il risultato nello stack

Es: Postscript, Java bytecode



# PILE

## SPECIFICA SINTATTICA

Tipi: *pila*, *boolean*, *tipoelem*

Operatori:

**CREAPILA:**  $( ) \rightarrow pila$

**PILAVUOTA:**  $(pila) \rightarrow boolean$

**LEGGIPILA:**  $(pila) \rightarrow tipoelem$

**FUORIPILA:**  $(pila) \rightarrow pila$

**INPILA:**  $(tipoelem, pila) \rightarrow pila$



# SPECIFICA SEMANTICA

## Tipi:

*pila=insieme delle sequenze  $P=a_1, a_2, \dots, a_n$  di elementi di tipo *tipoelem* gestita con accesso LIFO;*

*boolean=insieme dei valori di verità.*

## Operatori:

**CREAPILA**=P **EMPTY STACK**  
POST:  $P=\Lambda$  (LA SEQUENZA VUOTA)

**PILAVUOTA**(P)=b **EMPTY**  
POST: b=VERO SE  $P\equiv\Lambda$   
b=FALSO, ALTRIMENTI

**LEGGIPILA**(P)=a **TOP**  
PRE:  $P=a_1, a_2, \dots, a_n$   $n\geq 1$   
POST:  $a=a_1$

**FUORIPILA**(P)=P' **POP**  
PRE:  $P=a_1, a_2, \dots, a_n$   $n\geq 1$   
POST:  $P'=a_2, a_3, \dots, a_n$  SE  $n\geq 1$   
 $P'=\Lambda$  SE  $n=1$

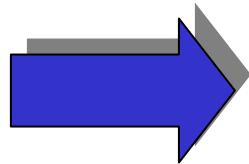
**INPILA**(a, P)= P' **PUSH**  
PRE:  $P=a_1, a_2, \dots, a_n$   $n\geq 0$   
POST:  $P'=a, a_1, a_2, \dots, a_n$



## REALIZZAZIONI

LA PILA È' UN CASO PARTICOLARE DI LISTA E OGNI REALIZZAZIONE DESCRITTA PER LA LISTA FUNZIONA ANCHE PER LA PILA. POSSIAMO DEFINIRE LA CORRISPONDENZA TRA GLI OPERATORI.

CREAPILA  
PILAVUOTA(P)  
LEGGIPILA(P)  
FUORIPILA(P)  
INPILA(a,P)



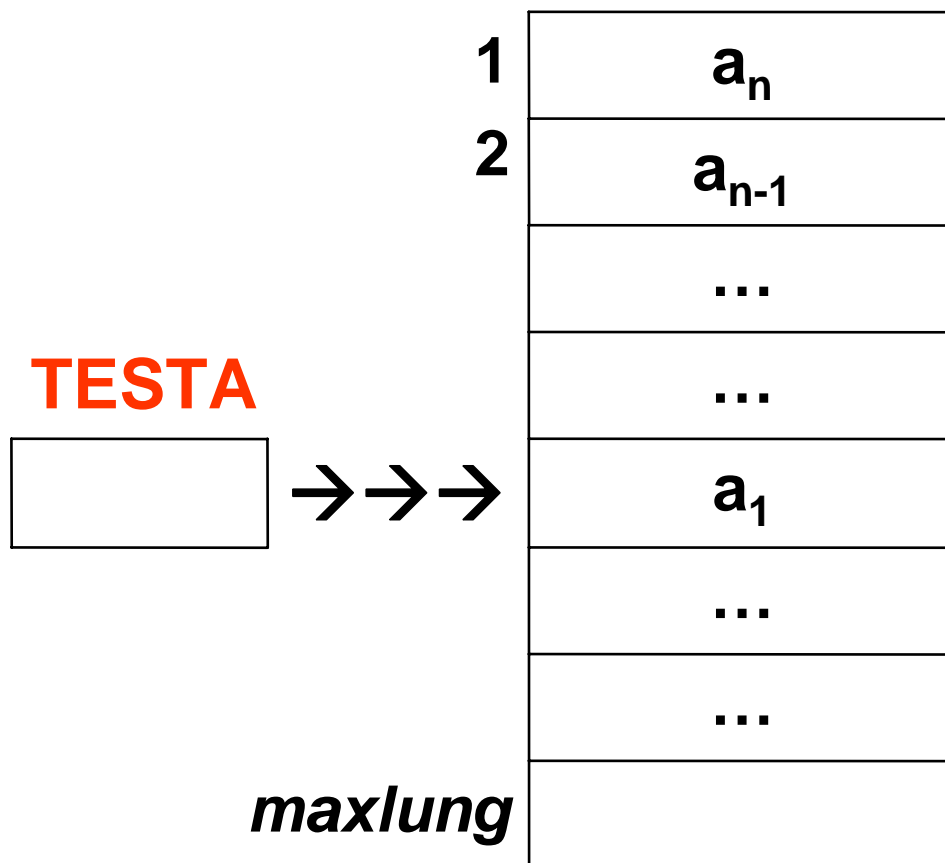
CREALISTA  
LISTAVUOTA(P)  
LEGGILISTA(PRIMOLISTA(P),P)  
CANCLISTA(PRIMOLISTA(P),P)  
INSLISTA(a,PRIMOLISTA(P),P)

Pensando invece a realizzarla direttamente, si fa riferimento a due rappresentazioni possibili della Pila: **rappresentazione sequenziale** (utilizzando il vettore) e **rappresentazione collegata**. Quest'ultima rappresentazione può essere realizzata con vettore o con puntatori. Nel seguito verrà considerata solo quella realizzata con puntatori.



## REALIZZAZIONE CON VETTORE

VANNO MEMORIZZATI GLI  $n$  ELEMENTI DELLA PILA, **IN ORDINE INVERSO**, NELLE PRIME  $n$  POSIZIONI DEL VETTORE, MANTENENDO UN CURSORE ALLA TESTA DELLA PILA.



## NOTE ALLA REALIZZAZIONE CON VETTORE

**CON QUESTA REALIZZAZIONE, OGNI OPERATORE RICHIEDE TEMPO COSTANTE PER ESSERE ESEGUITO.**

**LA RAPPRESENTAZIONE MEDIANTE ARRAY PRESENTA DUE SVANTAGGI:**

- RICHIEDE DI DETERMINARE A PRIORI UN LIMITE AL NUMERO MASSIMO DI ELEMENTI DELLA PILA;**
- LO SPAZIO DI MEMORIA UTILIZZATO È INDIPENDENTE DAL NUMERO EFFETTIVO DI ELEMENTI.**

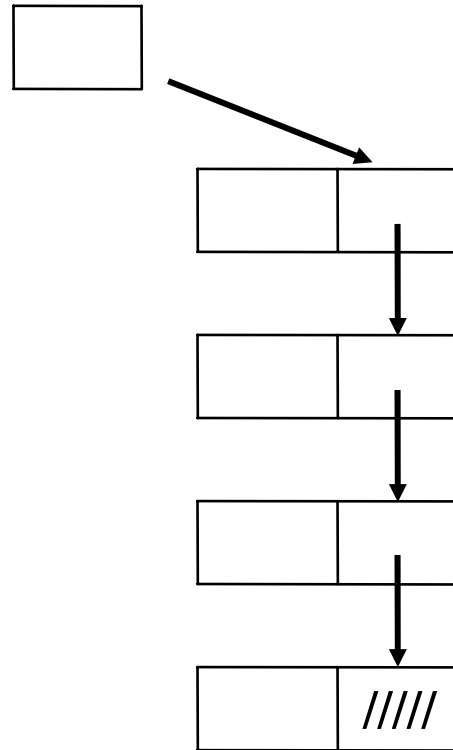
**PER CONTRO:**

**+ AL CONTRARIO DELLE LISTE, GLI INSERIMENTI E LE CANCELLAZIONI, NON RICHIEDONO SPOSTAMENTI PERCHÉ EFFETTUATI AD UNA ESTREMITÀ DELL'ARRAY.**



# RAPPRESENTAZIONE COLLEGATA: REALIZZAZIONE CON PUNTATORI

**CI RIFERIAMO ALLA PILA CON UN PUNTATORE ALLA CELLA CHE SI TROVA IN CIMA.**





## PILE E PROCEDURE RICORSIVE

UNA DELLE APPLICAZIONI PIÙ INTERESSANTI DELLE PILE RIGUARDA L'ESECUZIONE DI PROGRAMMI RICORSIVI.

L'ESECUZIONE DI UNA PROCEDURA RICORSIVA PREVEDE IL SALVATAGGIO DEI DATI SU CUI LAVORA LA PROCEDURA AL MOMENTO DELLA CHIAMATA RICORSIVA. TALI DATI VENGONO “RIPRISTINATI” QUANDO LA COMPUTAZIONE “INTERNA” TERMINA (MECCANISMO **LIFO**).

LE DIVERSE CHIAMATE ATTIVE SONO ORGANIZZATE IN UNA PILA. LA CHIAMATA PIÙ RECENTE È QUELLA CHE SI CONCLUDE PER PRIMA.

NELLA PILA VANNO SALVATI I PARAMETRI (E LE EVENTUALI VARIABILI LOCALI), IL PUNTO DI RITORNO, CIOÈ L'ETICHETTA DELLA ISTRUZIONE DA CUI RIPARTIRE AL TERMINE DELLA COMPUTAZIONE “INTERNA”.



**GRAZIE ALLE PILE È SEMPRE POSSIBILE, DATO UN PROGRAMMA RICORSIVO, TRASFORMARLO IN UNO ITERATIVO.**

**COME È NOTO, I PROGRAMMI RICORSIVI CORRISPONDONO A METODI SOLUTIVI PARTICOLARMENTE ADATTI A PROBLEMI PER I QUALI:**

- LA SOLUZIONE DEL PROBLEMA DI RANGO  $N$  È DEFINIBILE IN TERMINI DELLA SOLUZIONE DEL PROBLEMA DI RANGO INFERIORE A  $N$ ;**
- È DEFINIBILE UNA SOLUZIONE PER ASSIOMA SUL PROBLEMA DI RANGO MINIMO (1 O 0).**

### **ESEMPIO**

**LA SUCCESSIONE DI FIBONACCI È DEFINITA COME UNA SUCCESSIONE IL CUI  $K$ -ESIMO ELEMENTO È UGUALE ALLA SOMMA DEI DUE CHE LO PRECEDONO:**

**FIB(1)=1      FIB(2)=1**

***base***

***livello    assiomatico    o    di***

**FIB(K)=FIB(K-1)+FIB(K-2) PER  $K>2$**



**LA SEQUENZA DI FIBONACCI SI PUÒ CREARE MEDIANTE UNA FUNZIONE RICORSIVA CHE CALCOLA IL *K*-ESIMO NUMERO DELLA SUCCESSIONE.**

```
FIB(intero k) → intero  
  if (k=1) or (k=2) then  
    f ← 1 else  
    f ← FIB(k-1)+FIB(k-2); chi amata ricorsi va  
    FIB ← f  
  return FIB
```

### **GLI STATI DELLA PILA per K=4**

- 1 FIB(4)
- 2 FIB(3), FIB(4)
- 3 FIB(2)=1, FIB(3), FIB(4)
- 4 FIB(3), FIB(4) → RIS. PARZ.=1
- 5 FIB(1)=1, FIB(3), FIB(4) → RIS. PARZ.=1
- 6 FIB(3)=2, FIB(4) → RIS. PARZ.=2
- 7 FIB(4)
- 8 FIB(2)=1, FIB(4) → RIS. PARZ.=2
- 9 FIB(4)=3



LA **RICORSIONE** PUÒ ESSERE USATA PER FORMALIZZARE UN'AMPIA CLASSE DI **STRUTTURE** DI DATI CHE MOSTRANO CARATTERISTICHE **RICORSIVE** NELLA STRUTTURA.

IN GENERALE, SE INTENDIAMO PER **SEQUENZA** DI DATI UN AGGREGATO IN CUI SIA RICONOSCIBILE UN PRIMO ELEMENTO, UN SECONDO... UN SUCCESSIVO, POSSIAMO DEFINIRLA RICORSIVAMENTE COME:

- UN AGGREGATO DI DATI EVENTUALMENTE VUOTO;  
*livello assiomatico*
- UN AGGREGATO NON VUOTO, IN CUI È INDIVIDUATO UN PRIMO ELEMENTO CHE INSIEME AI SUCCESSIVI COSTITUISCE ANCORA UNA SEQUENZA.

SE INDICHIAMO LA SEQUENZA DI  $n$  ELEMENTI CON  $S_n = a_1, a_2, \dots, a_n$  ALLORA  $S_n$  SARÀ:

$$\left\{ \begin{array}{ll} \text{PER } N=0 & S_0 = \{ \} \\ \text{PER } N>0 & \{S_{n-1}, A_n\} \end{array} \right. \quad \textit{livello assiomatico}$$



## PER TRASFORMARE UNA PROCEDURA RICORSIVA IN UNA ITERATIVA BISOGNA:

- A) CREARE UNA PILA DOPO IL BEGIN INIZIALE;
- B) SOSTITUIRE OGNI CHIAMATA RICORSIVA CON UNA SEQUENZA DI ISTRUZIONI CHE:
- ***SALVANO NELLA PILA I VALORI DEI PARAMETRI DELLE VARIABILI LOCALI E L'ETICHETTA DELLA ISTRUZIONE SEGUENTE ALLA CHIAMATA RICORSIVA;***
  - ***ASSEGNANO AI PARAMETRI GLI OPPORTUNI VALORI;***
  - ***EFFETTUANO UN SALTO ALL'ISTRUZIONE CHE SEGUE LA CREAZIONE DELLA PILA.***
- C) INTRODURRE PRIMA DELL'END FINALE ISTRUZIONI CHE, NEL CASO LA PILA NON SIA VUOTA, ESTRAGGONO DALLA PILA I VALORI SALVATI E SALTANO ALLA ISTRUZIONE LA CUI ETICHETTA È UGUALE AL PUNTO DI RITORNO.



***I PARAMETRI SI INTENDONO PASSATI PER VALORE.***

**QUANDO PARAMETRI E VARIABILI LOCALI SONO DI TIPO DIVERSO NELLA PILA È CONVENIENTE MEMORIZZARE UN**

***RECORD DI ATTIVAZIONE***

**CHE CONTIENE LO STATO DELLA COMPUTAZIONE SOSPESA.**



## **ESERCIZIO**

**I DATI RELATIVI ALLE RICHIESTE DI INTERVENTO AD UN CALL CENTER, DOPO CHE QUESTE SONO STATE ESAUDITE, SONO MEMORIZZATI IN UNA PILA.**

**SI VOGLIONO EVIDENZIARE RICHIESTE DI DISTURBO PROVENIENTI DA UNO SPECIFICO UTENTE.**

**SCRIVERE UN PROGRAMMA CHE RICERCA I DATI RELATIVI AD UNA SPECIFICA RICHIESTA (UTENTE) E, SE PRESENTE, LA CANCELLA DALLA PILA RICOMPATTANDO LA STRUTTURA.**



# CODE

UNA **CODA** E' UN TIPO ASTRATTO CHE CONSENTE DI RAPPRESENTARE UNA SEQUENZA DI ELEMENTI CON ACCESSO AGLI ESTREMI: E' POSSIBILE AGGIUNGERE ELEMENTI AD UN ESTREMO (“**IL FONDO**”) E TOGLIERE ELEMENTI DALL'ALTRO ESTREMO (“**LA TESTA**”).

Detta anche **Queue**, è un insieme dinamico in cui l'elemento rimosso dall'operazione di cancellazione è predeterminato:

*“quello che per più tempo è rimasto nell'insieme”*  
politica “**first in, first out**” (**FIFO**)





# CODE

**E' PARTICOLARMENTE ADATTA A RAPPRESENTARE SEQUENZE NELLE QUALI L'ELEMENTO VIENE ELABORATO SECONDO L'ORDINE DI ARRIVO**

Possibili utilizzi:

Nei sistemi operativi, i processi in attesa di utilizzare una risorsa vengono gestiti tramite una coda (lista d'attesa, insieme di dispositivi in attesa di assegnazione di risorse, etc.)

La politica FIFO è fair



# CODE

Le operazioni definibili su una coda sono le seguenti :

- ☐ Inserimento di un elemento in fondo alla coda (*enqueue*)
- ☐ Estrazione dell'elemento in testa alla coda (*dequeue*)
- ☐ Lettura dell'elemento in testa alla coda, senza eliminarlo (*peek*)
- ☐ Controllo se la coda è vuota oppure no (*isEmpty*)

E talvolta

- ☐ Svuotamento della coda (*clear*)



# CODE

COME SEMPRE CI ATTENIAMO AD UN'ALGEBRA  
“LEGGERA”

SPECIFICA SINTATTICA

TIPI: **coda**, **boolean**, **tipoelem**

Operatori:

<b>CREACODA :</b>	<b>()</b>	<b>→ coda</b>
<b>CODAVUOTA :</b>	<b>(coda)</b>	<b>→ boolean</b>
<b>LEGGICODA :</b>	<b>(coda)</b>	<b>→ tipoelem</b>
<b>FUORICODA :</b>	<b>(coda)</b>	<b>→ coda</b>
<b>INCODA :</b>	<b>(tipoelem, coda)</b>	<b>→ coda</b>



# SPECIFICA SEMANTICA

Tipi: **CODA**: insieme delle sequenze  $Q = a_1, a_2, \dots, a_n$  di elementi di tipo **TIPOELEM**;

**BOOLEAN**: insieme dei valori di verità;

Operatori:

**CREACODA** =  $Q'$

Post:  $Q' = \Lambda$  sequenza vuota

**CODAVUOTA**( $Q$ ) =  $b$

Post:  $b = \text{VERO}$  SE  $Q = \Lambda$

$b = \text{FALSO}$  ALTRIMENTI

**LEGGICODA**( $Q$ ) =  $a$

Pre:  $Q = a_1, a_2, \dots, a_n$  E  $n \geq 1$

Post:  $a = a_1$



## **FUORICODA(Q) = Q'**

**Pre:      $Q = a_1, a_2, \dots, a_n$      $E \ n \geq 1$**

**Post:     $Q' = a_2, a_3, \dots, a_n$     $SE \ n > 1$**

**$Q' = \Lambda$     $SE \ n = 1$**

## **INCODA (a,Q) = Q'**

**Pre:      $Q = a_1, a_2, \dots, a_n$     $E \ n \geq 0$**

**Post:     $Q' = a_1, a_2, \dots, a_n, a$**

## **COME RAPPRESENTARE LE CODE**

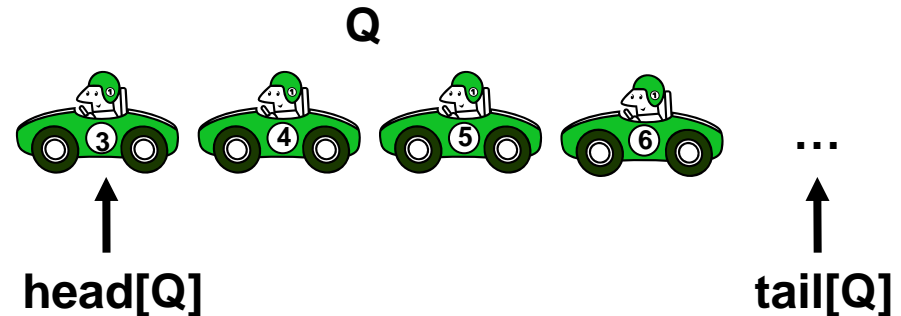
**IN GENERALE LE POSSIBILI RAPPRESENTAZIONI DELLE CODE SONO  
ANALOGHE A QUELLE DELLE PILE CON L'ATTENZIONE CHE E'  
CONVENIENTE CONSENTIRE L'ACCESSO SIA ALL'ELEMENTO INSERITO  
PER PRIMO SIA ALL'ELEMENTO INSERITO PER ULTIMO.**



# Coda con vettore

Operazioni:

- **INCODA(a,Q)** (*inserisci*)
- **FUORICODA(Q)** (*elimina, rimuovi*)



Una coda con al più  $n$  elementi può essere rappresentata da un vettore  $Q[1, \dots, n]$ . Le variabili  $head[Q]$  e  $tail[Q]$  puntano alla testa della coda e alla posizione dove il nuovo elemento sarà inserito.



# Coda con vettore

INCODA(a,Q)

1.  $Q[\text{tail}[Q]] \leftarrow a$
2. *if*  $\text{tail}[Q] = \text{lunghezza}[Q]$
3.     *then*  $\text{tail}[Q] \leftarrow 1$
4.     *else*  $\text{tail}[Q] \leftarrow \text{tail}[Q] + 1$

Inizialmente si avrà

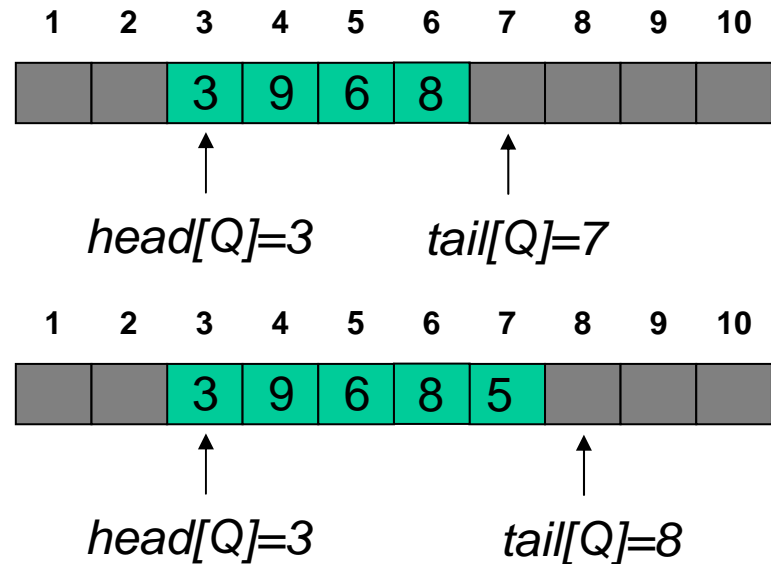
$\text{head}[Q] = \text{tail}[Q] = 1$ .

Quando  $\text{head}[Q] = \text{tail}[Q]$  la coda è vuota ( $\text{lunghezza} = 0$ ).

Un tentativo di prendere un elemento dalla coda genererà una situazione di “underflow”.

Quando  $\text{head}[Q] = \text{tail}[Q] + 1$  la coda è piena. Un tentativo di inserire un elemento dalla coda genererà una situazione di “overflow”.

INCODA(5,Q)

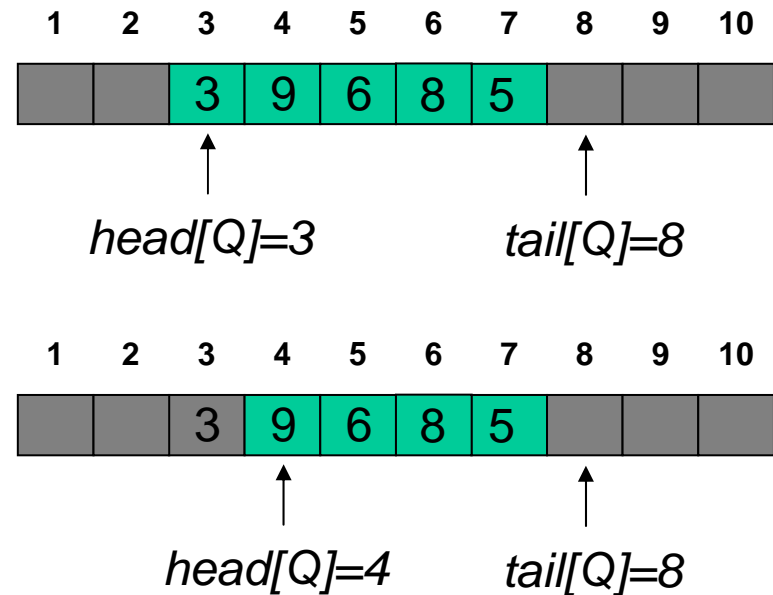


# Coda con vettore

FUORICODA(Q)

1. *if*  $head[Q] = length[Q]$
2.     *then*  $head[Q] \leftarrow 1$
3.     *else*  $head[Q] \leftarrow head[Q] + 1$
4.     *return*

FUORICODA(Q)



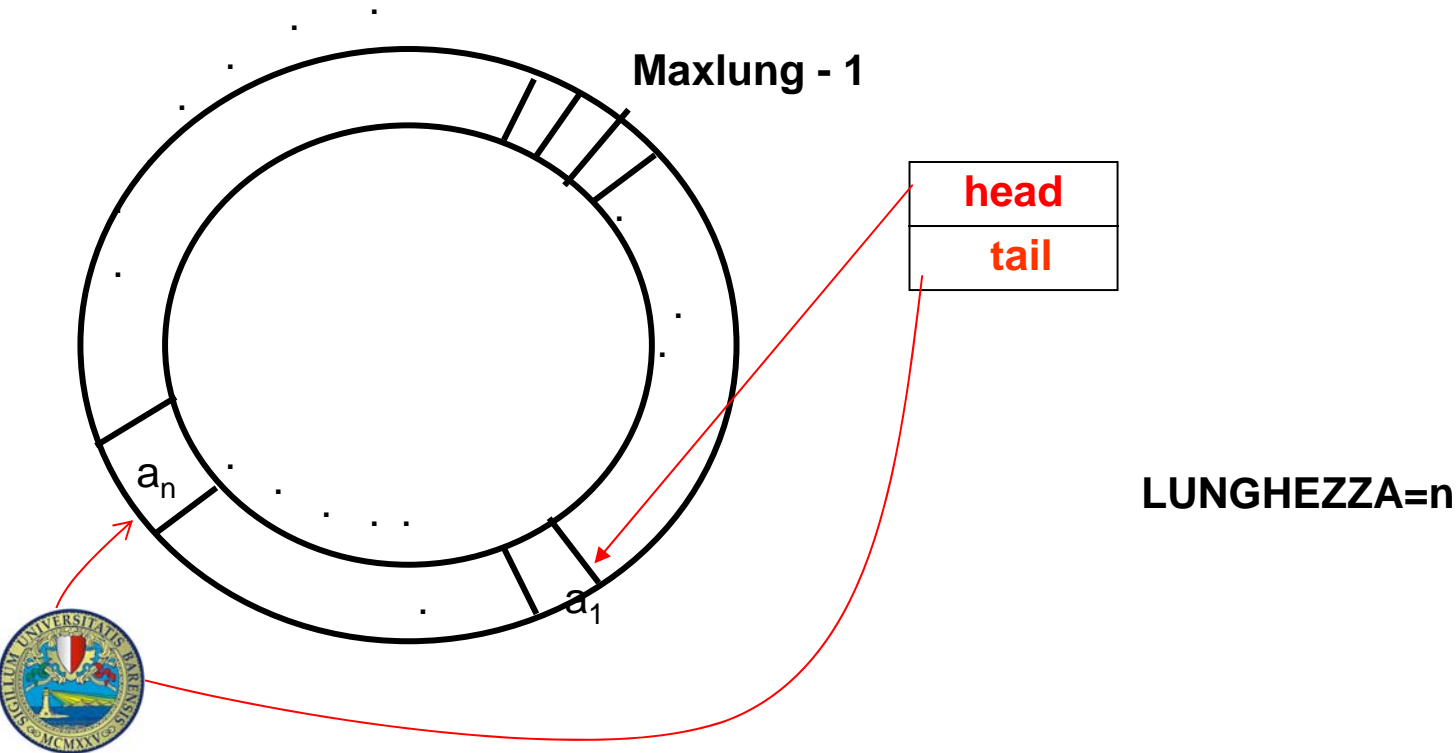
Le operazioni INCODA e FUORICODA richiedono tempo  $O(1)$ .





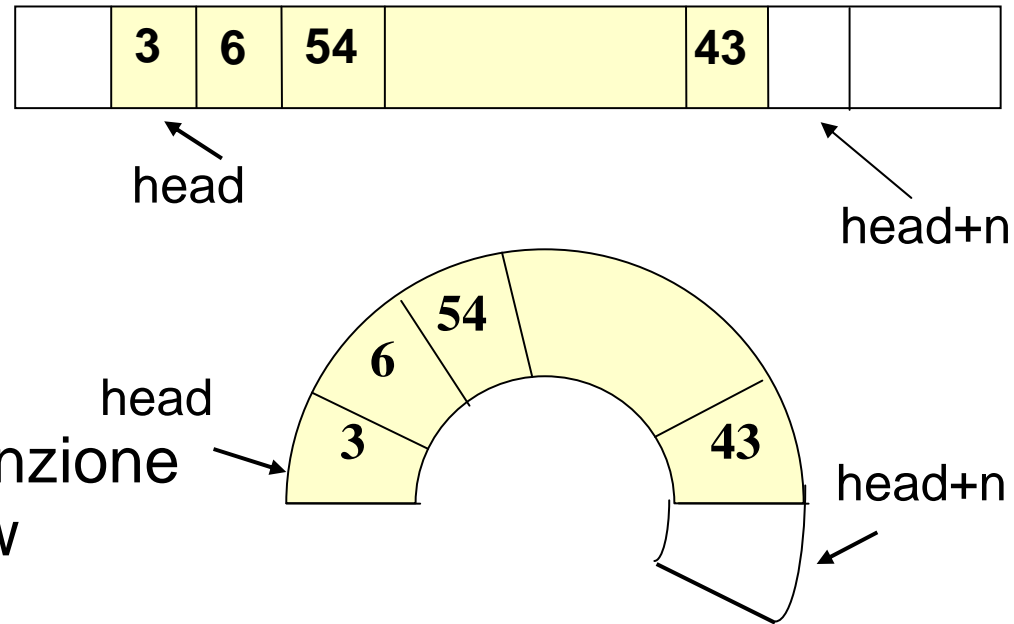
# Coda con vettore circolare

Spesso si usa un vettore circolare inteso come un array di  $\text{maxlung}$  elementi, nel quale si considera l'elemento con indice 0 come successore di quello con indice  $\text{maxlung} - 1$ . Si utilizzano due variabili **head** e **tail**. Il valore di **head** indica la posizione in cui è memorizzato il primo elemento mentre **tail** si riferisce all'ultimo elemento inserito. E' utile eventualmente definire la lunghezza della coda.

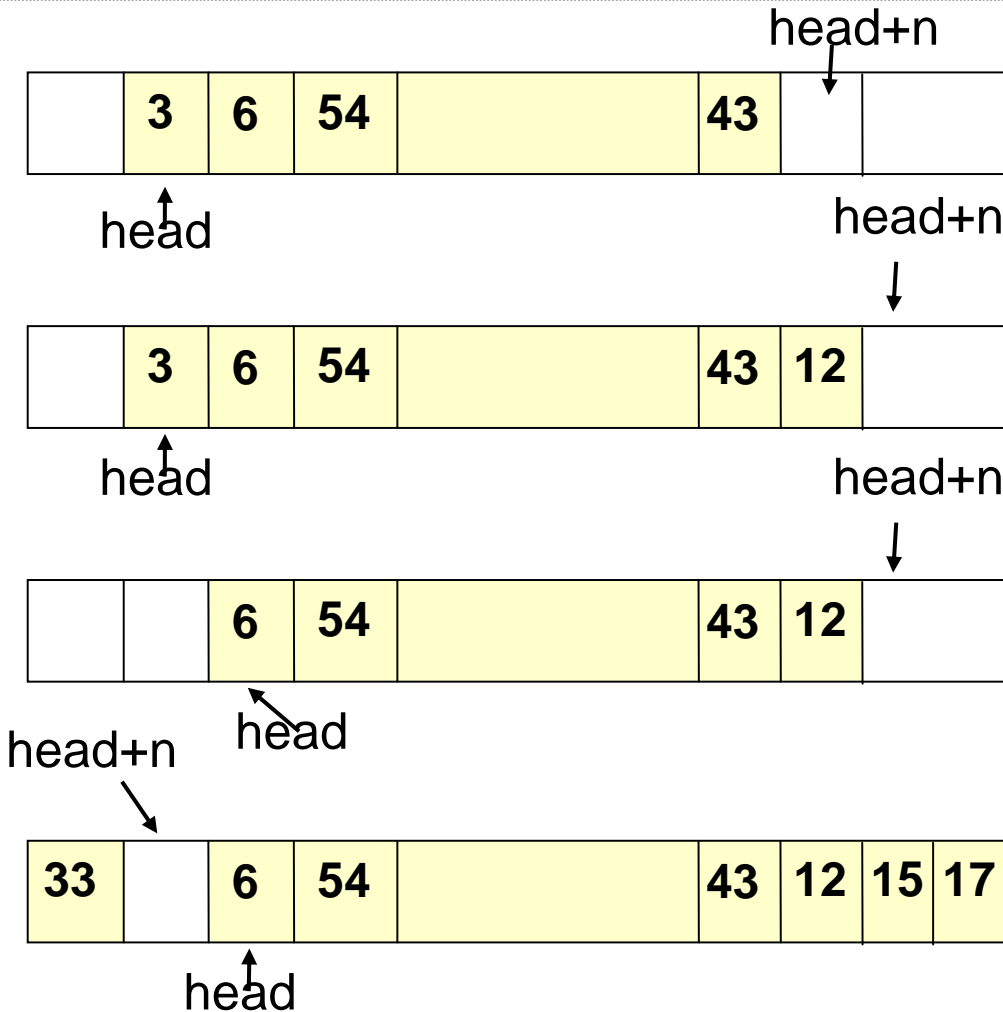


# Coda: Realizzazione tramite vettore circolare

- ♦ La “finestra” dell’array occupata dalla coda si sposta lungo l’array!
- ♦ Dettagli implementativi
  - ♦ L’array circolare può essere implementato con un’operazione di modulo
  - ♦ Bisogna prestare attenzione ai problemi di overflow (buffer pieno)



# Coda: Realizzazione tramite vettore circolare



INCODA(Q,12)

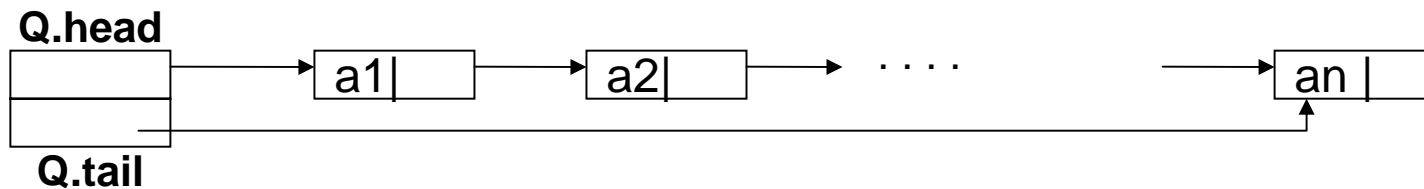
FUORICODA() → 3

INCODA(15,Q)

INCODA(17,Q)

INCODA(33,Q)





## REALIZZAZIONE CON PUNTATORI

LA CODA E' REALIZZATA CON  $n$  CELLE, LA PRIMA DELLE QUALI E' INDIRIZZATA DA UN PUNTATORE "HEAD" E L'ULTIMA DA UN PUNTATORE "TAIL".

LA CODA VUOTA E' INDIVIDUATA DAL VALORE NULLO Nil DEL PUNTATORE DI TESTA.

NELLA REALIZZAZIONE MONODIREZIONALE CON SENTINELLA LA VARIANTE PREVEDE UN ATOMO FITTIZIO PUNTATO DAL PUNTATORE TESTA CHE PUNTA AL VERO PRIMO ELEMENTO DELLA CODA.



# **ESERCIZI SU CODE E PILE**

# PROBLEMA

**SI VUOLE REALIZZARE UN PROGRAMMA CHE PRENDE UNA CODA DI INTERI E RESTITUISCE UN'ALTRA CODA OTTENUTA DALLA PRIMA CONSIDERANDO SOLO VALORI POSITIVI.**

*Trascuriamo le dichiarative relative alla implementazione della coda*

```
ESTRAI (q: coda, q1: coda)
  CREACODA(q1)
  while not CODAVUOTA(q)
    e ← LEGGI CODA(q)
    if e > 0 then
      INCODA(e, q1)
  FUORI CODA(q)
```



# SE VOLESSIMO CONSERVARE LA CODA ORIGINALE DOVREMMO USARE UNA CODA AUSILIARIA.

ESTRAI 1 ( $q$ ,  $q1$ : coda)

*CREACODA*( $q1$ )

*CREACODA*( $qaux$ )

*while not CODAVUOTA*( $Q$ ) *do*

$e \leftarrow$  *LEGGI CODA*( $q$ )

*if*  $e > 0$  *then*

*INCODA* ( $e$ ,  $q1$ )

*FUORI CODA*( $q$ )

*INCODA*( $e$ ,  $qaux$ )

*CREACODA*( $q$ )

*while not CODAVUOTA*( $qaux$ ) *do*

$e \leftarrow$  *LEGGI CODA*( $qaux$ )

*INCODA* ( $e$ ,  $q$ )

*FUORI CODA* ( $qaux$ )

RIPRISTINO DI Q



## ESEMPIO

### *Aritmetica postfissa*

LE ESPRESSIONI ARITMETICHE SONO SCRITTE IN NOTAZIONE **INFISSA**, CIOE' I SIMBOLI DELLE OPERAZIONI APPAIONO TRA GLI OGGETTI SU CUI OPERANO.

NELLA NOTAZIONE **POSTFISSA** GLI OPERATORI SI PONGONO **DOPO** GLI OGGETTI SU CUI OPERANO

$$35 + 17 * (40 - 9) - 7$$

$$35 \ 17 \ 40 \ 9 \ - \ * \ + \ 7 \ -$$

QUESTA NOTAZIONE E' DEFINITA POLACCA (SI DEVE AL MATEMATICO LUKASIEWICZ)





# Aritmetica postfissa

## Definizione

UNA NOTAZIONE POLACCA E' UNA QUALUNQUE SERIE DI OPERANDI ARITMETICI  $x, y, \dots$  E OPERATORI BINARI  $op (+, -, *, /)$  CHE SI PUO' FORMARE MEDIANTE LE REGOLE SEGUENTI:

- **OGNI OPERANDO  $x$  E' UNA NOTAZIONE POLACCA**
- **SE  $p1$  E  $p2$  SONO NOTAZIONI POLACCHE ALLORA  $p1p2 op$  E' UNA NOTAZIONE POLACCA**

ES:

$48+63$  corrisponde a  $p1 \quad 48 \quad 63 \quad +$

$52*4$  corrisponde a  $p2 \quad 52 \quad 4 \quad *$

$p1 \quad p2$  – è una notazione polacca



# ***Aritmetica postfissa***

## **LE REGOLE PER CALCOLARE L'ESPRESSIONE POSTFISSA**

- **SCANDISCI L'ESPRESSIONE DA SINISTRA A DESTRA FINO A CHE RAGGIUNGI IL PRIMO OPERATORE**
- **APPLICA L'OPERATORE AI DUE OPERANDI CHE SONO ALLA SUA SINISTRA, OTTIENI IL RISULTATO CHE SOSTITUIRAI NELL'ESPRESSIONE AL POSTO DI OPERANDI E OPERATORE**



$$35 + 17 * (40 - 9) - 7$$

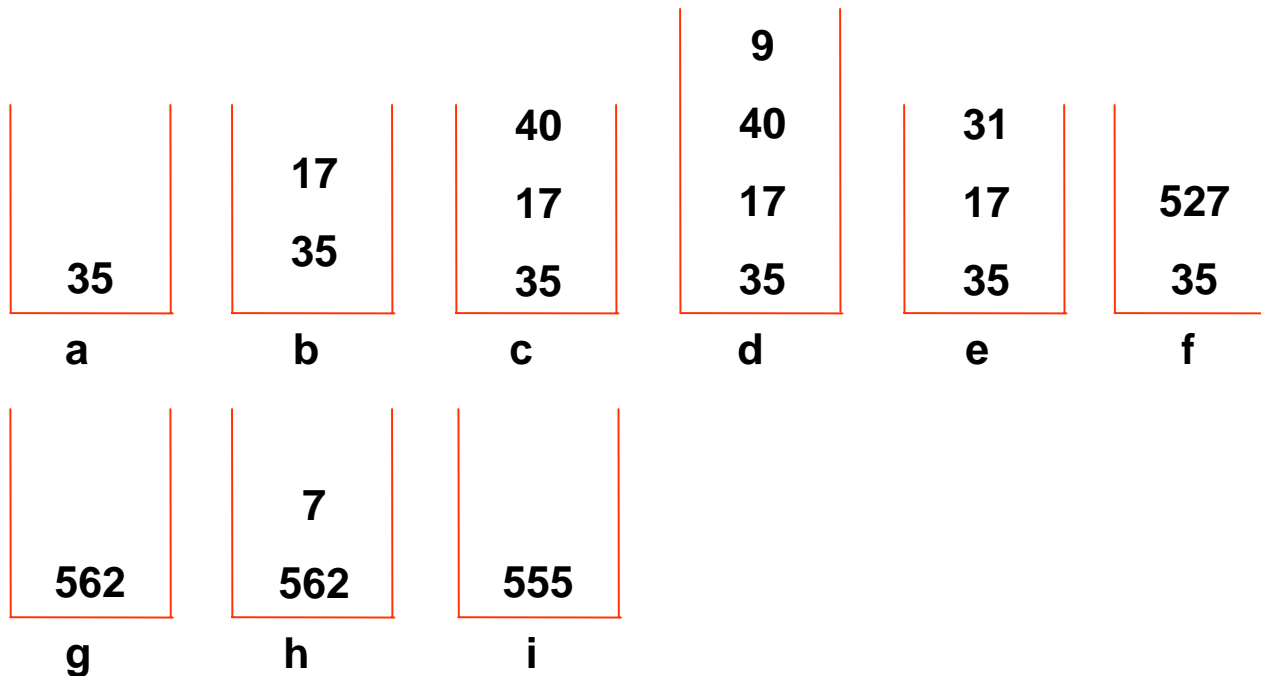
$$35 \ 17 \ 40 \ 9 \ - \ * \ + \ 7 \ -$$

- 35 17 40 9 - \* + 7 -
- 35 17 31 \* + 7 -
- 35 527 + 7 -
- 562 7 -
- 555



# ALGORITMO PER VALUTARE UNA ESPRESSIONE IN NOTAZIONE POLACCA

- 1. SCANDIRE L'ESPRESSIONE DA SINISTRA A DESTRA.**  
APPENA E' RAGGIUNTO UN OPERANDO COMPI IL PASSO 2.  
APPENA E' RAGGIUNTO UN OPERATORE COMPI IL PASSO 3.
- 2. IN-PILA L'OPERANDO NELLA PILA DEGLI OPERANDI**
- 3. RIMUOVI DALLA PILA I PRIMI DUE OPERANDI, APPLICA L'OPERATORE A QUESTI E IN-PILA IL RISULTATO IN CIMA ALLA PILA.**



IL PROGRAMMA DI SEGUITO REALIZZA L'ALGORITMO, MA RIMANDA ALCUNE FUNZIONALITA' (VERIFICA CHE IL SIMBOLO SIA UN NUMERO) AD UN ULTERIORE SFORZO DI PROGRAMMAZIONE.

SI DISPONE DI UNA LISTA DI SIMBOLI E SI USA UNA PILA PER LA VALUTAZIONE DELLA NOTAZIONE POLACCA MEDIANTE PILA DI NUMERI REALI.

```
VALUTA_POLACCA (post: lista simboli) → real  
  CREAPILA (val)  
  while not LISTAVUOTA (post) do  
    t ← LEGGILISTA(post)  
    if ( tnum = t E' UN NUMERO) then  
      INPILA (tnum, val)  
    else  
      numtop ← LEGGIPILA (val)  
      FUORIPILA(val)  
      numsuc ← LEGGIPILA (val)  
      FUORIPILA(val)  
      RIS ← numsuc ' t' numtop  
      INPILA (ris, val)  
  return LEGGIPILA(val)
```



**DISPORRE DI UN MODO PER CALCOLARE ESPRESSIONI POSTFISSE SERVE A POCO SE NON SI DISPONE DI UN METODO CHE SIA IN GRADO DI CONVERTIRE LA LISTA DI SIMBOLI DI UNA ESPRESSIONE INFISSA IN QUELLI DELLA CORRISPONDENTE ESPRESSIONE POSTFISSA. IN QUESTO CASO E' CONVENIENTE UTILIZZARE UNA CODA COME RISULTATO DELLA CONVERSIONE.**



**UTILIZZIAMO UNA PILA PER MEMORIZZARE I SIMBOLI DEGLI OPERATORI “IN SOSPESO” E UNA CODA PER IMMAGAZZINARE LA ESPRESSIONE POSTFISSA CHE VIENE COSTRUITA MAN MANO. AD OGNI OPERATORE NELLA ESPRESSIONE INFISSA VIENE ASSEGNATO UN ORDINE DI PRECEDENZA (MOLTIPLICAZIONE E DIVISIONE PRECEDENZA MASSIMA). ATTRIBUIAMO ALLA PARENTESI ( LA PRECEDENZA MINIMA.**



LISTA SIMBOLI	PILA	CODA
$35 + 17 * (40 - 9) - 7$	$\emptyset$	$\emptyset$
$+ 17 * (40 - 9) - 7$	$\emptyset$	35
$17 * (40 - 9) - 7$	+	35
$* (40 - 9) - 7$	+	17 35
$(40 - 9) - 7$	+ *	17 35
$40 - 9) - 7$	+ * (	17 35
$- 9) - 7$	+ * (	40 17 35
$9) - 7$	+ * ( -	40 17 35
$) - 7$	+ * ( -	9 40 17 35
$- 7$	+ * (	- 9 40 17 35
7	-	+*- 9 40 17 35
$\emptyset$	-	7+*- 9 40 17 35
$\emptyset$	$\emptyset$	-7+*-9 40 17 35





TRASFERISCI(s: pila, c: coda)  
top\_elem ← LEGGIPILA(s)  
FUORIPILA(s)  
INCODA(top\_elem, c)

CONVERTI(infiessa: lista, coda\_post: coda per riferimento)

CREAPILA(pila\_op)  
CREACODA(coda\_post)  
while not LISTAVUOTA(infiessa) do  
    t ← LEGGLISTA(infiessa)  
    if (t è un numero ) then  
        INCODA(t, coda\_post)  
    else if PILAVUOTA(pila\_op) then  
        INPILA(t, pila\_op)  
    else if (t = "(" ) then  
        INPILA(t, pila\_op)  
    else if (t = ")" ) then  
        top\_elem ← LEGGIPILA(pila\_op)  
        while (top\_elem ≠ "(" ) do  
            TRASFERISCI(pila\_op, coda\_post)  
            FUORIPILA(pila\_op)  
    else  
        while (PRIORITA' t <= PRIORITA' top\_elem ) do  
            TRASFERISCI(pila\_op, coda\_post)  
            INPILA(t, pila\_op)  
while not FINEPILA(pila\_op) do  
    TRASFERISCI(pila\_op, coda\_post)

