Pila

[Specifica sintattica](#)

[Specifica semantica](#)

[Realizzazioni](#)

[Realizzazione con vettore](#)

[Realizzazione con puntatori](#)

[Pile e procedure ricorsive](#)
- [La torre di Hanoi](#)
- [La torre di Hanoi: versione iterativa](#)

[Valutazione di un'espressione postfissa](#)

[Conversione di un'espressione infissa in postfissa](#)

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, detta **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.

Specifica sintattica

Tipi: `pila, boolean, tipoelem`
Operatori:
- `creapila: () → pila`
- `pilavuota: (pila) → boolean`
- `leggipila: (pila) → tipoelem`
- `fuoripila: (pila) → pila`
- `inpila: (tipoelem, pila) → pila`

Specifica semantica

Tipi:
- `pila`: insieme delle sequenze $P = < a_1, a_2, ..., a_n >$ con $n \geq 0$, di elementi di tipo tipoelem gestita con accesso LIFO
- `boolean`: insieme dei valori di verità

Operatori:
- `creapila = p`
  - Post: $p = <>$
- `pilavuota(p) = b`
  - Post: $b = true$ se $p = <>$; $b = false$ altrimenti

- `leggipila(p) = a`
  - Pre: $p = < a_1, a_2, \ldots, a_n >, n \geq 1$
  - Post: $a = a_1$
- `fuoripila(p) = p'`
  - Pre: $p = < a_1, a_2, \ldots, a_n >, n \geq 1$
  - Post: $p' = < a_2, a_3, \ldots, a_n >$ se $n > 1$

    $p' = < >$ se $n = 1$
- `inpila(a, p) = p'`
  - Pre: $p = < a_1, a_2, \ldots, a_n >, n \geq 0$
  - Post: $p' = < a, a_1, a_2, \ldots, a_n >$ se $n \geq 1$

    $p' = < a >$ se $n = 0$

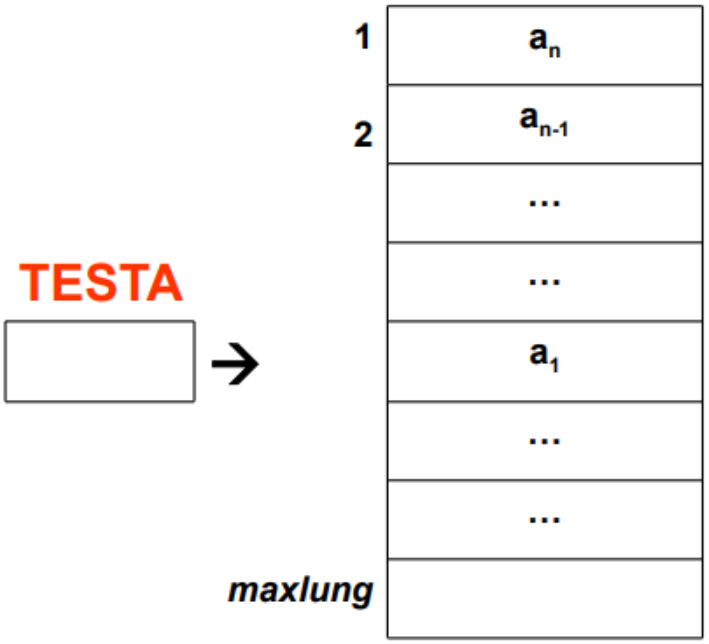
Realizzazioni

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()` → **`crealista()`**
- `pilavuota(p)` → **`listavuota(p)`**
- `leggipila(p)` → **`leggilista`**`(`**`primolista(p),`** `p)`
- `fuoripila(p)` → **`canclista`**`(`**`primolista(p),`** `p)`
- `inpila(a, p)` → **`inslista`**`(a,` **`primolista(p)`**`)`

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.



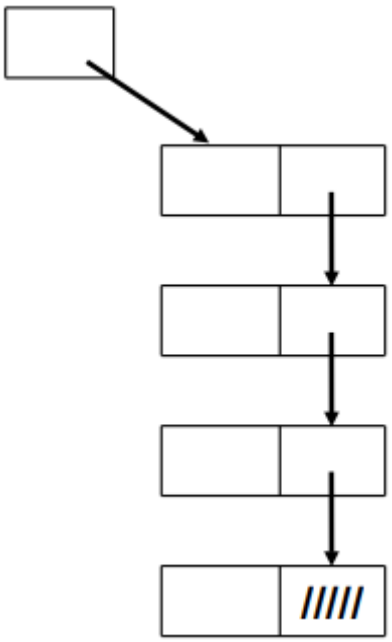
Con questa realizzazione, ogni operatore richiede **tempo costante** per essere eseguito. La rappresentazione mediante array presenta due **svantaggi**:

1. Richiede di determinare a priori un limite al **numero massimo di elementi** della pila;
2. Lo spazio di memoria utilizzato è **indipendente** dal numero effettivo di elementi.

Tuttavia, al contrario delle liste, gli inserimenti e le cancellazioni **non richiedono spostamenti** perché effettuati ad una estremità dell'array.

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**.

La torre di Hanoi

In un monastero tibetano ci sono n dischi d'oro, forati al centro, tutti di diametro diverso. I dischi sono infilati in un piolo verticale, accatastati per diametro decrescente a partire dal basso. I monaci devono spostare tutti i dischi dal piolo in cui si trovano in un altro piolo, formando una catasta identica a quella di partenza, spostando un disco alla volta senza mai sovrapporre un disco più grande su un disco più piccolo. È possibile usare un terzo piolo di appoggio per effettuare trasferimenti.

Regole:

- Si può spostare un solo disco per volta da un piolo ad un altro;
- Dopo ogni mossa, per ogni piolo, i dischi devono avere diametro decrescente dal basso verso l'alto.


1.1.1.1. trasferisci un disco da 1 a 3


1.1.2. trasferisci un disco da 1 a 2


1.1.3.1. trasferisci un disco da 3 a 2


1.2. trasferisci un disco da 1 a 3


1.3.1.1. trasferisci un disco da 2 a 1


1.3.2. trasferisci un disco da 2 a 3


1.3.3.1. trasferisci un disco da 1 a 3

Per dire “spostare n dischi dal piolo sorgente al piolo destinazione usando il piolo intermedio” usiamo la **notazione**:

$$p(n,\ sorgente,\ destinazione,\ intermedio)$$

La **formalizzazione** della torre di Hanoi di ordine n, cioè spostare n dischi dal piolo 1 al piolo 3, usando il piolo 2, può essere:

$$p(n,\ 1,\ 3,\ 2)$$

- Se $n = 1$ basta spostare il disco da 1 a 3.
- Se $n > 1$
  - $p(n - 1,\ 1,\ 2,\ 3)$ sposta $n - 1$ dischi da origine (1) a intermedio (2) usando destinazione (3)
  - Trasferisci da origine (1) a destinazione (3) il disco rimanente
  - $p(n - 1,\ 2,\ 3,\ 1)$ sposta $n - 1$ dischi da intermedio (2) a destinazione (3) usando origine (1)

Nel caso di $n = 3$ i sottoproblemi sono:

1. $p(3,\ 1,\ 3,\ 2)$
   a. $p(2,\ 1,\ 2,\ 3)$

i. $p(1,\ 1,\ 3,\ 2)$
    1. trasferisci da 1 a 3
ii. trasferisci da 1 a 2
iii. $p(1,\ 3,\ 2,\ 1)$
    1. trasferisci da 3 a 2

b. trasferisci da 3 a 2
c. $p(2,\ 2,\ 3,\ 1)$

2. …

La torre di Hanoi: versione iterativa

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:
    i) **Salvano** nella pila i valori dei parametri delle variabili locali e l'etichetta della istruzione seguente alla chiamata ricorsiva
    ii) **Assegnano** ai parametri gli opportuni valori
        1) Viene seguito il codice relativo alle sequenze di istruzioni per questa chiamata
    iii) 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.

Valutazione di un'espressione postfissa

Utilizzo di una pila per valutare operazioni aritmetiche del tipo:

$$5\ *\ (((9\ +\ 8)\ *\ (4\ *\ 6))\ +\ 7)$$

Il calcolo richiede di memorizzare i risultati intermedi. Una pila è il meccanismo ideale per memorizzare risultati intermedi in questi calcoli.
Iniziamo con il problema più semplice:

- L'espressione da valutare è in forma **postfissa**;
- Ogni **operatore** appare dopo i suoi due argomenti;
- Ogni **espressione** aritmetica può essere sempre riorganizzata in forma postfissa

$$5\ 9\ 8\ +\ 4\ 6\ *\ *\ 7\ +\ *$$

Procedendo da sinistra a destra nell'espressione:

- Se incontriamo un **numero**, lo **inseriamo** (push) nella pila;
- Se incontriamo un **operatore**, inseriamo nella pila il **risultato** dell'applicazione dell'operatore ai due operandi che si trovano sulla cima della pila.

Ogni operatore verrà interpretato come il comando di "estrarre" (pop) gli operandi dalla cima della pila, eseguire l'operazione e reinserire il risultato nella pila.

Conversione di un’espressione infissa in postfissa

Le pile consentono anche di convertire le espressioni **infisse**, con parentesi che racchiudono ogni operazione, in espressioni **postfisse**.
Al fine di eseguire tale operazione, inseriamo gli operatori in una pila, mentre gli operandi sono dati direttamente in output. Ogni parentesi chiusa indica che entrambi gli argomenti dell’ultima operazione sono stati dati in output, perciò l’operatore può essere estratto dalla pila e dato anch’esso in output.
Per convertire l’espressione:

$$(5 * (((9 + 8) * (4 * 6)) + 7))$$

nella sua forma postfissa

$$5\,9\,8\,+\,4\,6\,*\,*\,7\,+\,*$$

Procediamo da sinistra a destra:

- Se incontriamo un **numero**, lo scriviamo in **output**;
- Se incontriamo una **parentesi aperta**, la **ignoriamo**;
- Se incontriamo un **operatore**, lo inseriamo nella **pila**;
- Se incontriamo una **parentesi chiusa**, scriviamo l’operatore che sta in **cima** alla pila in output.