

# Algoritmi e Strutture Dati

## Strutture di dati

Alberto Montresor

Università di Trento

2018/10/19

This work is licensed under a Creative Commons  
Attribution-ShareAlike 4.0 International License.



# Sommario

- 1 Strutture dati astratte
  - Definizioni
  - Sequenza
  - Insiemi
  - Dizionari
  - Alberi e grafi
- 2 Implementazione strutture dati elementari
  - Lista
  - Pila
  - Coda

# Introduzione

## Dato

In un linguaggio di programmazione, un dato è un valore che una variabile può assumere

## Tipo di dato astratto

Un modello matematico, dato da una collezione di valori e un insieme di operazioni ammesse su questi valori

## Tipi di dato primitivi

- Forniti direttamente dal linguaggio
- Esempi: int (+, -, \*, /, %), boolean (!, &&, ||)

# Tipi di dati

## “Specifica” e “implementazione” di un tipo di dato astratto

- **Specifica**: “manuale d’uso”, nasconde i dettagli implementativi all’utente
- **Implementazione**: realizzazione vera e propria

## Esempi

Specifica	Implementazione
Numeri reali	IEEE-754
Pile	Pile basate su vettori Pile basate su puntatori
Code	Code basate su vettori circolari Code basate su puntatori

# Strutture di dati

## Strutture di dati

Le strutture di dati sono collezioni di dati, caratterizzate più dall'organizzazione della collezione piuttosto che dal tipo dei dati contenuti.

## Come caratterizzare le strutture dati

- un insieme di operatori che permettono di manipolare la struttura
- un modo sistematico di organizzare l'insieme dei dati

## Alcune tipologie di strutture di dati

- **Lineari** / **Non lineari** (presenza di una sequenza)
- **Statiche** / **Dinamiche** (variazione di dimensione, contenuto)
- **Omogenee** / **Disomogenee** (dati contenuti)

# Strutture di dati

Tipo	Java	C++	Python
Sequenze	<b>List, Queue, Deque</b> LinkedList, ArrayList, Stack, ArrayDeque	list, forward_list vector stack queue, deque	list tuple
Insiemi	<b>Set</b> TreeSet, HashSet, LinkedHashSet	set unordered_set	set, frozenset
Dizionari	<b>Map</b> HashTree, HashMap, LinkedHashMap	map unordered_map	dict
Alberi	-	-	-
Grafi	-	-	-

# Sequenza

## Sequenza

Una struttura dati **dinamica**, **lineare** che rappresenta una sequenza **ordinata** di valori, dove un valore può comparire più di una volta.

- L'ordine all'interno della sequenza è importante

## Operazioni ammesse

- Data la posizione, è possibile aggiungere / togliere elementi
  - $s = s_1, s_2, \dots, s_n$
  - l'elemento  $s_i$  è in posizione  $pos_i$
  - esistono le posizioni (fittizie)  $pos_0, pos_{n+1}$
- È possibile accedere direttamente alla testa/coda
- È possibile accedere sequenzialmente a tutti gli altri elementi

# Sequenza – Specifica

---

## SEQUENCE

---

% Restituisce **true** se la sequenza è vuota

**boolean** isEmpty()

% Restituisce **true** se  $p$  è uguale a  $pos_0$  oppure a  $pos_{n+1}$

**boolean** finished(POS  $p$ )

% Restituisce la posizione del primo elemento

POS head()

% Restituisce la posizione dell'ultimo elemento

POS tail()

% Restituisce la posizione dell'elemento che segue  $p$

POS next(POS  $p$ )

% Restituisce la posizione dell'elemento che precede  $p$

POS prev(POS  $p$ )

---

## Sequenza – Specifica

---

### SEQUENCE (continua)

---

% Inserisce l'elemento  $v$  di tipo ITEM nella posizione  $p$ .

% Restituisce la posizione del nuovo elemento, che diviene il predecessore di  $p$

**POS insert(POS  $p$ , ITEM  $v$ )**

% Rimuove l'elemento contenuto nella posizione  $p$ .

% Restituisce la posizione del successore di  $p$ ,

% che diviene successore del predecessore di  $p$

**POS remove(POS  $p$ )**

% Legge l'elemento di tipo ITEM contenuto nella posizione  $p$

**ITEM read(POS  $p$ )**

% Scrive l'elemento  $v$  di tipo ITEM nella posizione  $p$

**write(POS  $p$ , ITEM  $v$ )**

---

# Sequenza

## Java

```
List<String> lista = new LinkedList<String>();  
lista.add("two");  
lista.addFirst("one");  
lista.addLast("three");
```

**Result:** ['one', 'two', 'three']

## C++

```
std::list<int> lista;  
lista.push_front(2);  
lista.push_front(1);  
lista.push_back(3);
```

**Result:** [1,2,3]

## Python

```
lista = ["one", "three"]  
lista.insert(1, "two")
```

**Result:** [ 'one', 'two', 'three' ]

# Insiemi

## Insieme

Una struttura dati **dinamica**, **non lineare** che memorizza una **collezione non ordinata di elementi** senza valori ripetuti.

- L'ordinamento fra elementi è dato dall'eventuale relazione d'ordine definita sul tipo degli elementi stessi

## Operazioni ammesse

- Operazioni base:
  - inserimento
  - cancellazione
  - verifica contenimento
- Operazioni di ordinamento
  - Massimo
  - Minimo
- Operazioni insiemistiche:
  - unione
  - interesezione
  - differenza
- Iteratori:
  - **foreach**  $x \in S$  **do**

# Insiemi – Specifica

---

## SET

---

% Restituisce la cardinalità dell'insieme

**int** = *ize()*

% Restituisce **true** se  $x$  è contenuto nell'insieme

**boolean** *contains*(ITEM  $x$ )

% Inserisce  $x$  nell'insieme, se non già presente

**insert**(ITEM  $x$ )

% Rimuove  $x$  dall'insieme, se presente

**remove**(ITEM  $x$ )

% Restituisce un nuovo insieme che è l'unione di  $A$  e  $B$

**SET** *union*(SET  $A$ , SET  $B$ )

% Restituisce un nuovo insieme che è l'intersezione di  $A$  e  $B$

**SET** *intersection*(SET  $A$ , SET  $B$ )

% Restituisce un nuovo insieme che è la differenza di  $A$  e  $B$

**SET** *difference*(SET  $A$ , SET  $B$ )

---

# Insiemi

## Java

```
Set<String> docenti = new TreeSet<>();  
docenti.add("Alberto");  
docenti.add("Cristian");  
docenti.add("Alessio");
```

**Result:** { "Alberto", "Alessio", "Cristian" }

## C++

```
std::set<std::string> frutta;  
frutta.insert("mele");  
frutta.insert("pere");  
frutta.insert("banane");  
frutta.insert("mele");  
frutta.remove("mele")
```

**Result:** { "banane", "pere" }

## Python

```
items = { "rock", "paper",  
          "scissors", "rock" }  
print(items)  
print("Spock" in items)  
print("lizard" not in items)
```

**Result:**

```
{ "rock", "paper", "scissors" }  
False  
True
```

# Dizionari

## Dizionario

Struttura dati che rappresenta il concetto matematico di **relazione univoca**  $R : D \rightarrow C$ , o associazione chiave-valore.

- Insieme  $D$  è il **dominio** (elementi detti **chiavi**)
- Insieme  $C$  è il **codominio** (elementi detti **valori**)

## Operazioni ammesse

- Ottenere il valore associato ad una particolare chiave (se presente), o **nil** se assente
- Inserire una nuova associazione chiave-valore, cancellando eventuali associazioni precedenti per la stessa chiave
- Rimuovere un'associazione chiave-valore esistente

## Dizionari – Specifica

---

### DICTIONARY

---

% Restituisce il valore associato alla chiave  $k$  se presente, **nil**  
altrimenti

**ITEM lookup**(ITEM  $k$ )

% Associa il valore  $v$  alla chiave  $k$

**insert**(ITEM  $k$ , ITEM  $v$ )

% Rimuove l'associazione della chiave  $k$

**remove**(ITEM  $k$ )

---

# Array associativi, mappe e dizionari

## Java

```
Map<String, String> capoluoghi = new HashMap<>();
capoluoghi.put("Toscana", "Firenze");
capoluoghi.put("Lombardia", "Milano");
capoluoghi.put("Sardegna", "Cagliari");
```

## C++

```
std::map<std::string, int> wordcounts;
std::string s;

while (std::cin >> s && s != "end")
    ++wordcounts[s];
```

## Python

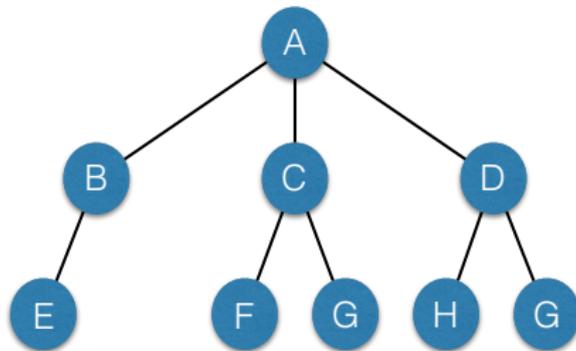
```
v = {}
v[10] = 5
v["alberto"] = 42
v[10]+v["alberto"]
```

**Result:** 47

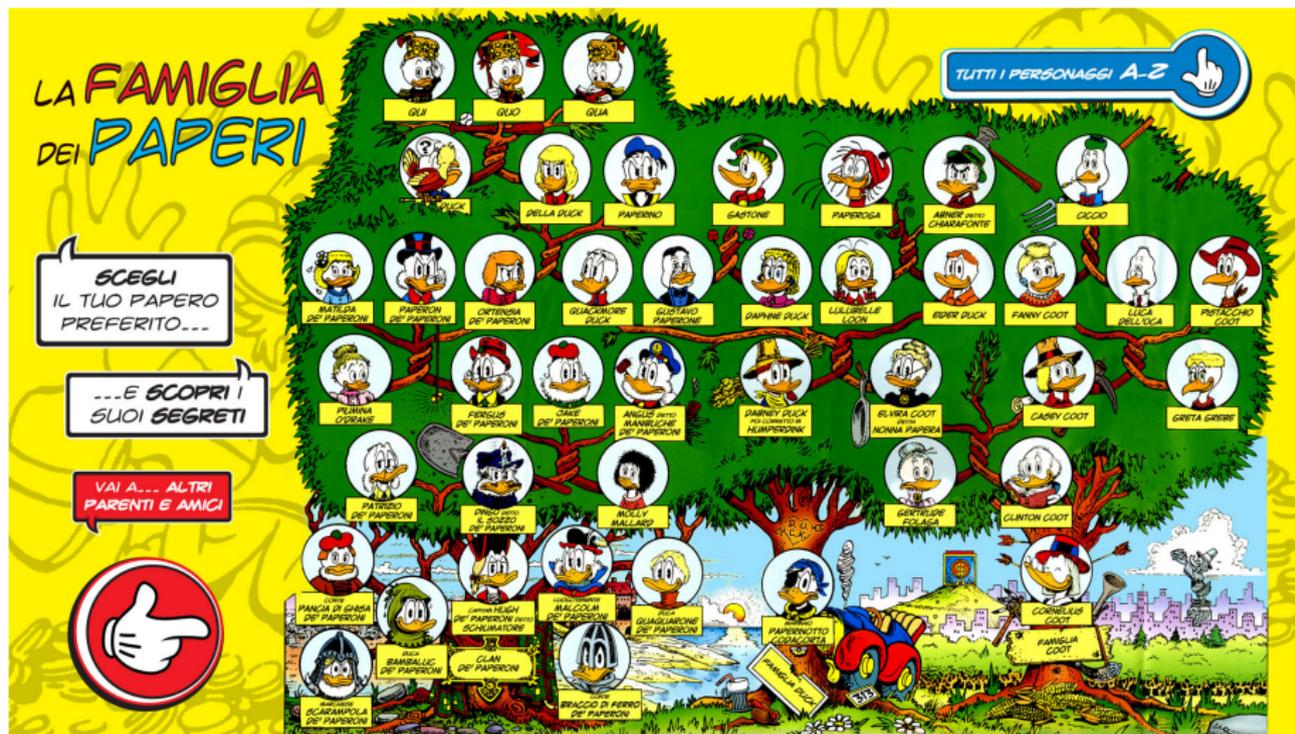
# Alberi e grafi

## Alberi ordinati

- Un **albero ordinato** è dato da un insieme finito di elementi detti **nodi**
- Uno di questi nodi è designato come **radice**
- I rimanenti nodi, se esistono sono partizionati in insiemi **ordinati** e **disgiunti**, anch'essi alberi ordinati



# Alberi e grafi

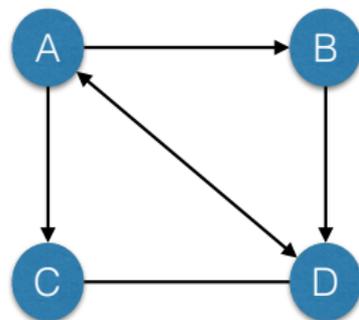


# Alberi e grafi

## Grafi

La struttura dati grafo è composta da:

- un insieme di elementi detti **nodi** o **vertici**
- un insieme di coppie (ordinate oppure no) di nodi detti **archi**



## Operazioni

- Tutte le operazioni su alberi e grafi ruotano attorno alla possibilità di effettuare **visite** su di essi
- Specifica completa più avanti

# Commenti

- Concetti di sequenza, insieme, dizionario sono collegati
  - Insieme delle chiavi / insieme dei valori
  - Scorrere la sequenza di tutte le chiavi
- Alcune realizzazioni sono "naturali"
  - Sequenza  $\leftrightarrow$  lista
  - Albero astratto  $\leftrightarrow$  albero basato su puntatori
- Esistono tuttavia realizzazioni alternative
  - Insieme come vettore booleano
  - Albero come vettore dei padri
- La scelta della struttura di dati ha riflessi sull'efficienza e sulle operazioni ammesse
  - Dizionario come hash table: lookup  $O(1)$ , ricerca minimo  $O(n)$
  - Dizionario come albero: lookup  $O(\log n)$ , ricerca minimo  $O(1)$

# Sommario

## 1 Strutture dati astratte

- Definizioni
- Sequenza
- Insiemi
- Dizionari
- Alberi e grafi

## 2 Implementazione strutture dati elementari

- Lista
- Pila
- Coda

# Lista

## Lista (Linked List)

Una sequenza di nodi, contenenti dati arbitrari e 1-2 puntatori all'elemento successivo e/o precedente.

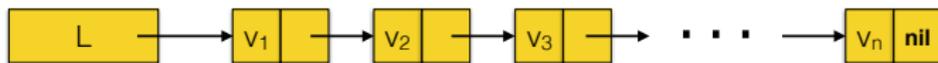
## Note

- Contiguità nella lista  $\nrightarrow$  contiguità nella memoria
- Tutte le operazioni hanno costo  $O(1)$

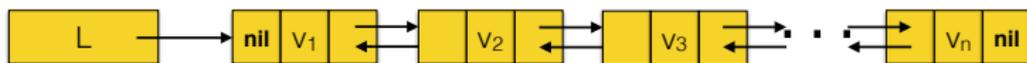
## Possibili implementazioni

- Bidirezionale / Monodirezionale
- Con sentinella / Senza sentinella
- Circolare / Non circolare

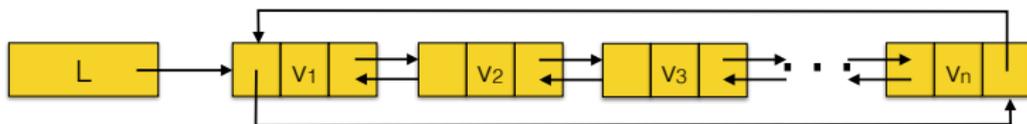
# Lista



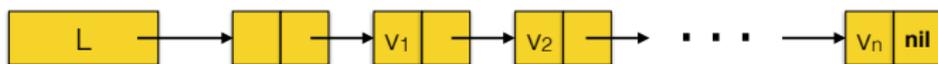
Monodirezionale



Bidirezionale



Bidirezionale circolare



Monodirezionale con sentinella

# Lista bidirezionale con sentinella

---

```

LIST
LIST pred
LIST succ
ITEM value

LIST List()
  LIST t = new LIST
  t.pred = t
  t.succ = t
  return t

boolean isEmpty()
  return pred = succ = this

POS head()
  return succ

POS tail()
  return pred

POS next(POS p)
  return p.succ

POS prev(POS p)
  return p.pred

% Predecessore
% Successore
% Elemento
boolean finished(POS p)
  return (p == this)

ITEM read(POS p)
  return p.value

write(POS p, ITEM v)
  p.value = v

POS insert(POS p, ITEM v)
  LIST t = List()
  t.value = v
  t.pred = p.pred
  p.pred.succ = t
  t.succ = p
  p.pred = t
  return t;

POS p)
  p.pred.succ = p.succ
  p.succ.pred = p.pred
  LIST t = p.succ
  delete p
  return t;

```

---

LIST e POS sono tipi equivalenti

## Lista bidirezionale senza sentinella – Java

```
class Pos {  
  
    Pos succ;           /** Next element of the list */  
    Pos pred;          /** Previous element of the list */  
    Object v;          /** Value */  
  
    Pos(Object v) {  
        succ = pred = null;  
        this.v = v;  
    }  
}
```

## Lista bidirezionale senza sentinella – Java

```
public class List {  
  
    private Pos head;      /** First element of the list */  
    private Pos tail;     /** Last element of the list */  
  
    public List() {  
        head = tail = null;  
    }  
  
    public Pos head()      { return head; }  
    public Pos tail()     { return tail; }  
    public boolean finished(Pos pos) { return pos == null; }  
    public boolean isEmpty() { return head == null; }  
    public Object read(Pos p) { return p.v; }  
    public void write(Pos p, Object v) { p.v = v; }
```

## Lista bidirezionale senza sentinella – Java

```
public Pos next(Pos pos) {  
    return (pos != null ? pos.succ : null);  
}
```

```
public Pos prev(Pos pos) {  
    return (pos != null ? pos.pred : null);  
}
```

```
public void remove(Pos pos) {  
    if (pos.pred == null)  
        head = pos.succ;  
    else  
        pos.pred.succ = pos.succ;  
    if (pos.succ == null)  
        tail = pos.pred;  
    else  
        pos.succ.pred = pos.pred;  
}
```

## Lista bidirezionale senza sentinella – Java

```
public Pos insert(Pos pos, Object v) {
    Pos t = new Pos(v);
    if (head == null) {
        head = tail = t; // Insert in a empty list
    } else if (pos == null) {
        t.pred = tail; // Insert at the end
        tail.succ = t;
        tail = t;
    } else {
        t.pred = pos.pred; // Insert in front of an existing position
        if (t.pred != null)
            t.pred.succ = t;
        else
            head = t;
        t.succ = pos;
        pos.pred = t;
    }
    return t;
}
```

## Liste

```
prev->next = toDelete->next;  
delete toDelete;
```

```
// if only forgetting were  
// this easy for me.
```



```
assert "It's going to be okay.";
```



# Pila

## Pila (Stack)

Una struttura dati dinamica, lineare in cui l'elemento rimosso dall'operazione di cancellazione è predeterminato: “quello che per meno tempo è rimasto nell'insieme” (**LIFO - Last-in, First-out**)

---

### STACK

---

% Restituisce **true** se la pila è vuota

**boolean** isEmpty()

% Inserisce *v* in cima alla pila

**push**(ITEM *v*)

% Estrae l'elemento in cima alla pila e lo restituisce al chiamante

**ITEM** pop()

% Legge l'elemento in cima alla pila

**ITEM** top()

---

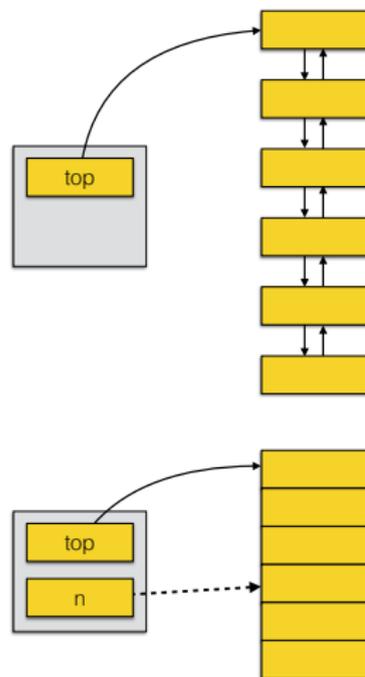
# Pila

## Possibili utilizzi

- Nei linguaggi con procedure:
  - gestione dei record di attivazione
- Nei linguaggi stack-oriented:
  - le operazioni prendono gli operandi dallo stack e inseriscono il risultato nello stack
  - Es: Postscript, Java bytecode

## Possibili implementazioni

- Tramite **liste bidirezionali**
  - puntatore all'elemento top
- Tramite **vettore**
  - dimensione limitata, overhead più basso



# Pila basata su vettore – Pseudocodice

---

## STACK

---

ITEM[] *A*                   % Elementi  
 int *n*                        % Cursore  
 int *m*                        % Dim. massima

STACK Stack(int *dim*)

```

STACK t = new STACK
t.A = new int[1...dim]
t.m = dim
t.n = 0
return t
  
```

ITEM top()

```

precondition: n > 0
return A[n]
  
```

boolean isEmpty()

```

return n = 0
  
```

ITEM pop()

```

precondition: n > 0
ITEM t = A[n]
n = n - 1
return t
  
```

push(ITEM *v*)

```

precondition: n < m
n = n + 1
A[n] = v
  
```

---

## Pila basata su vettore – Java

```
public class VectorStack implements Stack {  
  
    /** Vector containing the elements */  
    private Object[] A;  
  
    /** Number of elements in the stack */  
    private int n;  
  
    public VectorStack(int dim) {  
        n = 0;  
        A = new Object[dim];  
    }  
  
    public boolean isEmpty() {  
        return n==0;  
    }  
}
```

## Pila basata su vettore – Java

```
public Object top() {  
    if (n == 0)  
        throw new IllegalStateException("Stack is empty");  
    return A[n-1];  
}
```

```
public Object pop() {  
    if (n == 0)  
        throw new IllegalStateException("Stack is empty");  
    return A[--n];  
}
```

```
public void push(Object o) {  
    if (n == A.length)  
        throw new IllegalStateException("Stack is full");  
    A[n++] = o;  
}
```

```
}
```

# Coda



## Coda (Queue)

Una struttura dati dinamica, lineare in cui l'elemento rimosso dall'operazione di cancellazione è predeterminato: “quello che per più tempo è rimasto nell'insieme” (**FIFO - First-in, First-out**)

---

## QUEUE

---

% Restituisce **true** se la coda è vuota

**boolean** isEmpty()

% Inserisce *v* in fondo alla coda

**enqueue**(ITEM *v*)

% Estrae l'elemento in testa alla coda e lo restituisce al chiamante

**ITEM** dequeue()

% Legge l'elemento in testa alla coda

**ITEM** top()

---

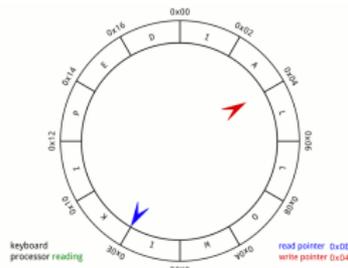
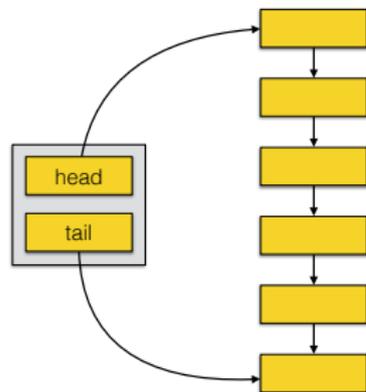
# Coda

## Possibili utilizzi

- Nei sistemi operativi, i processi in attesa di utilizzare una risorsa vengono gestiti tramite una coda
- La politica FIFO è **fair**

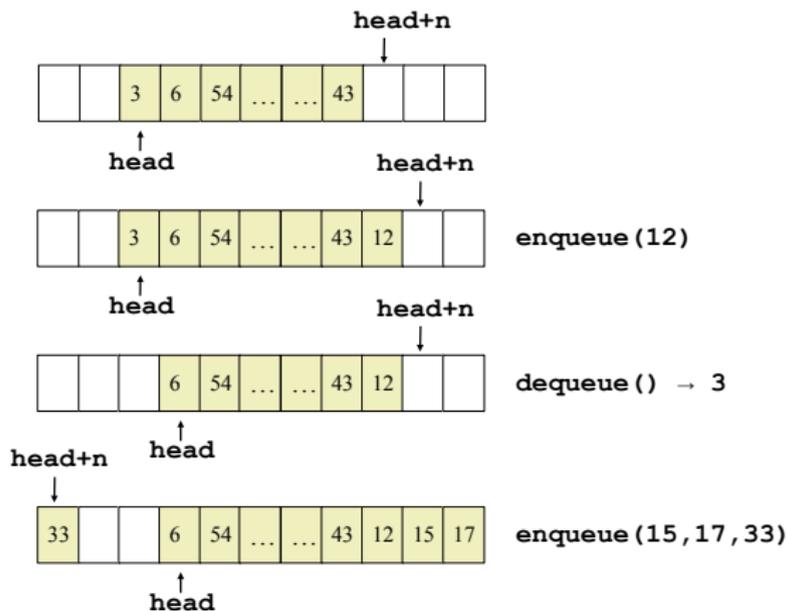
## Possibili implementazioni

- Tramite **liste monodirezionali**
  - puntatore head, per estrazione
  - puntatore tail, per inserimento
- Tramite **array circolari**
  - dimensione limitata, overhead più basso

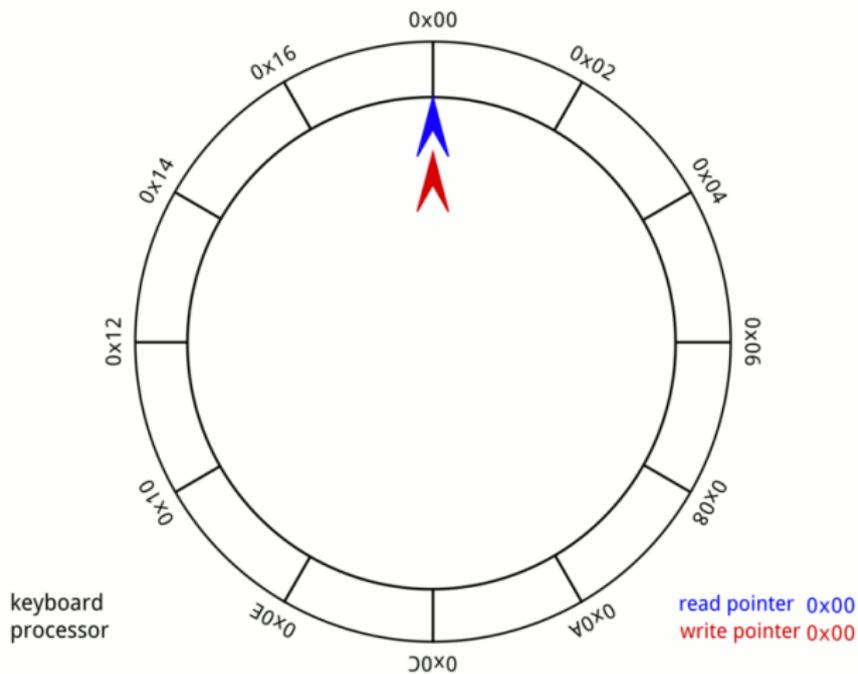


## Coda basata su vettore circolare

- La circolarità può essere implementata con l'operazione **modulo**
- Bisogna prestare attenzione ai problemi di **overflow** (buffer pieno)

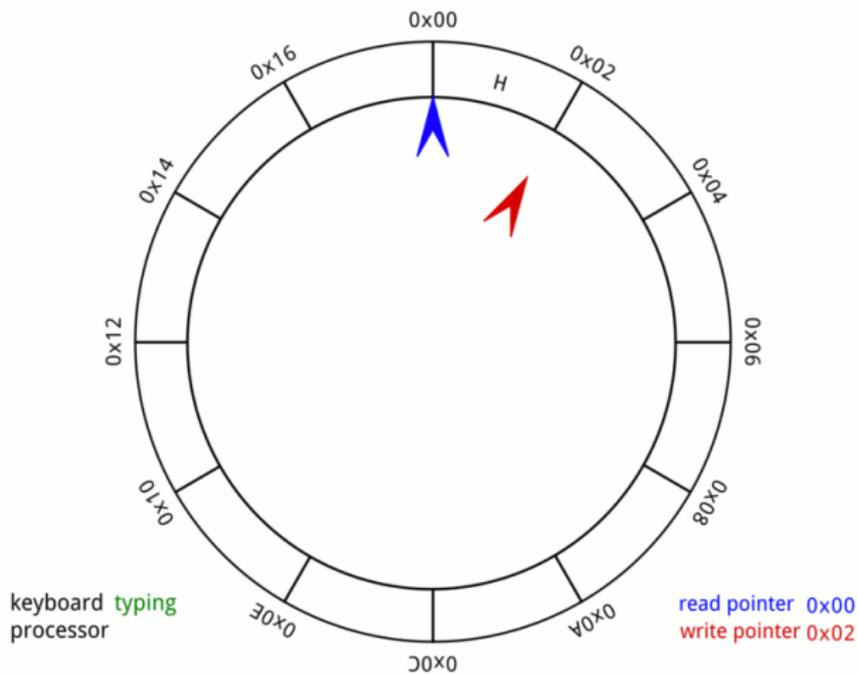


# Coda basata su vettore circolare



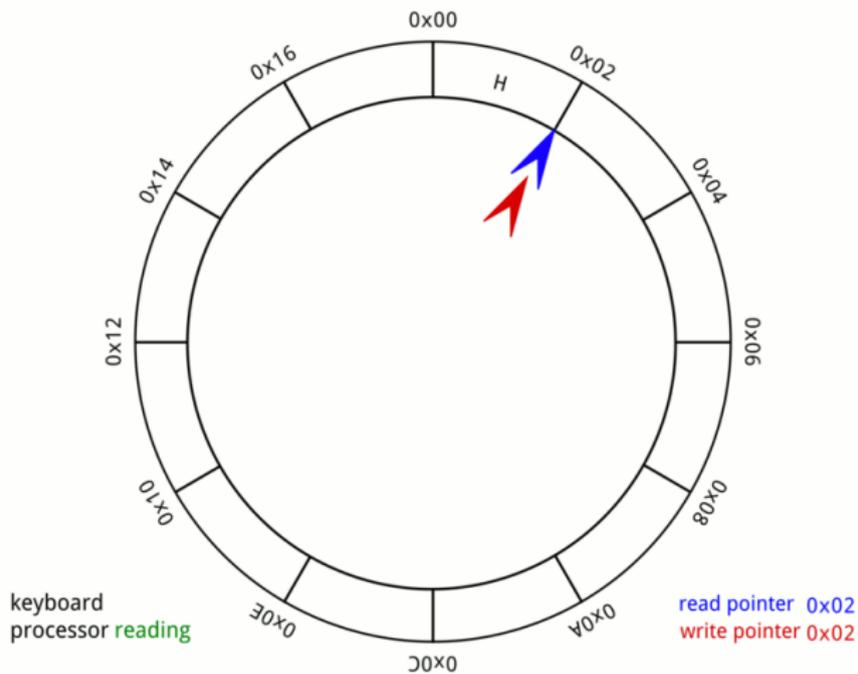
By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

# Coda basata su vettore circolare



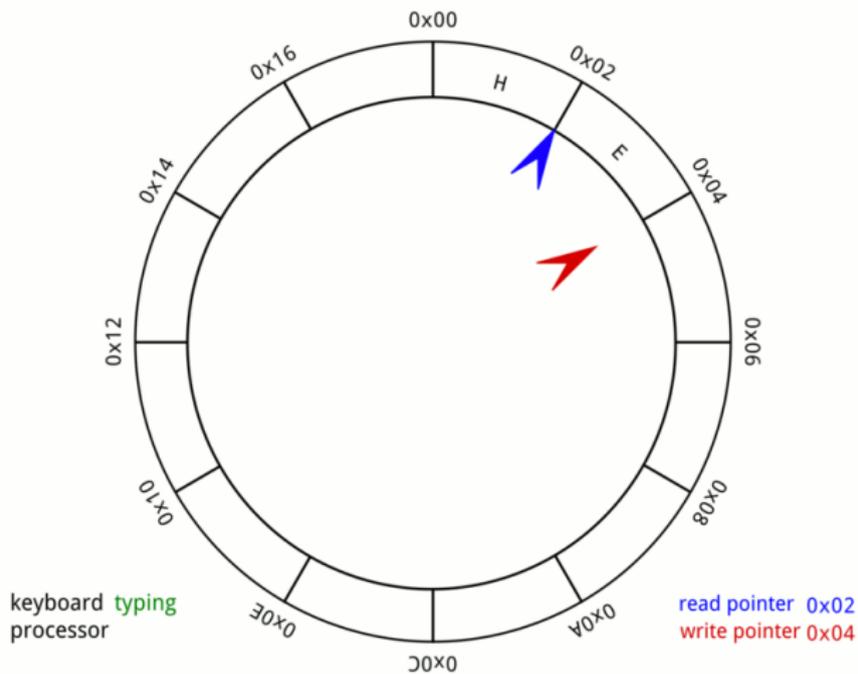
By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

# Coda basata su vettore circolare



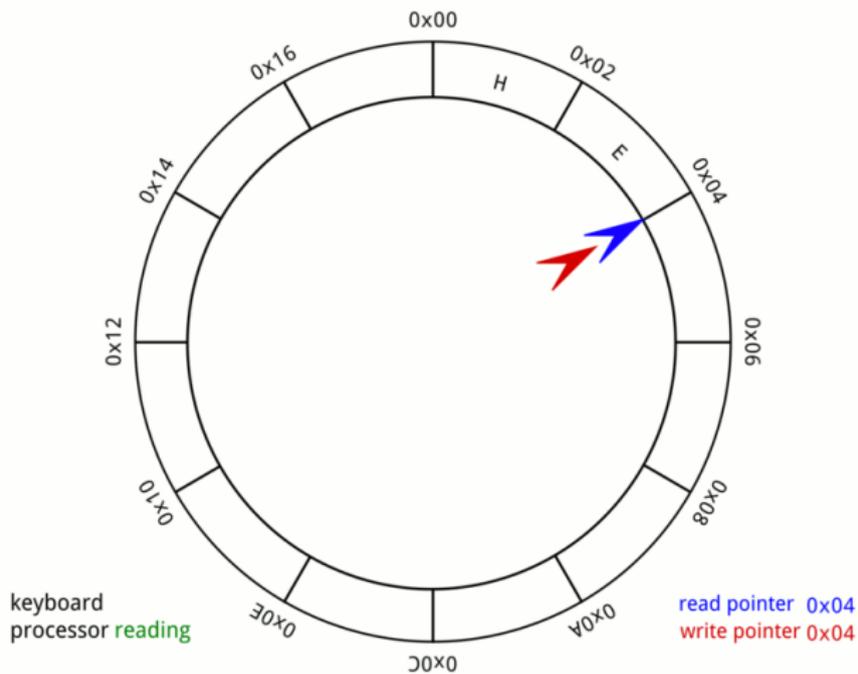
By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

# Coda basata su vettore circolare



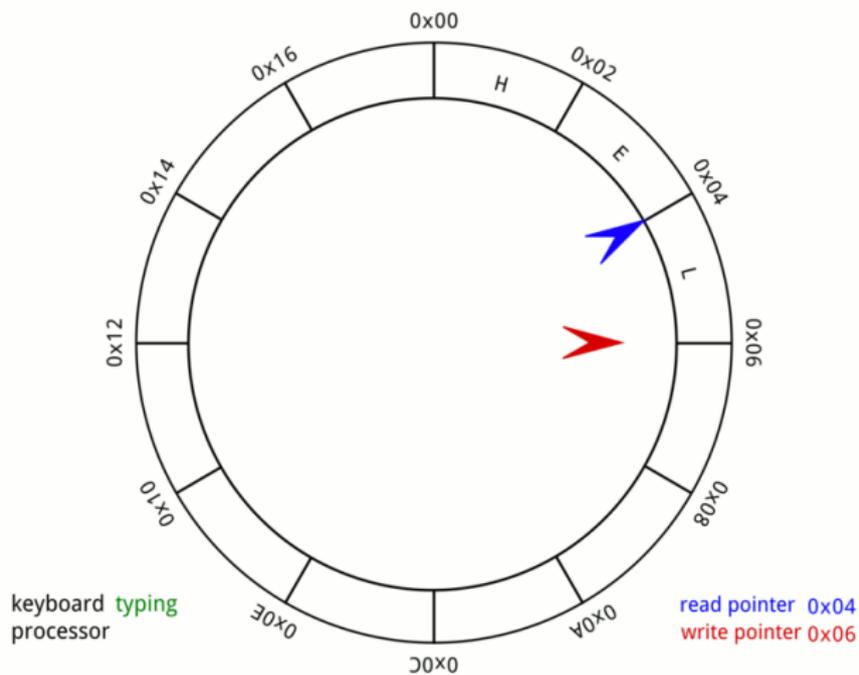
By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

# Coda basata su vettore circolare



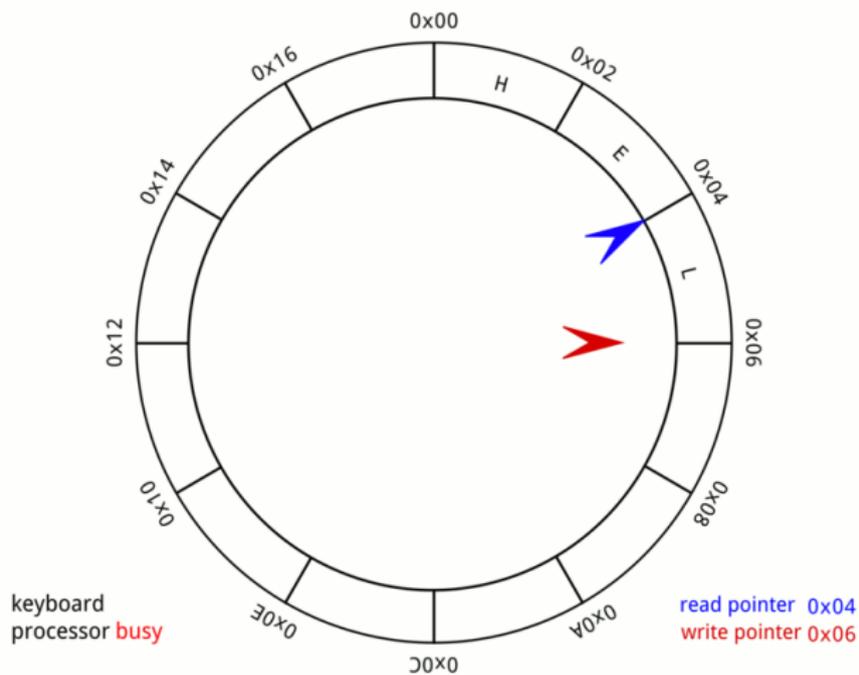
By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

# Coda basata su vettore circolare



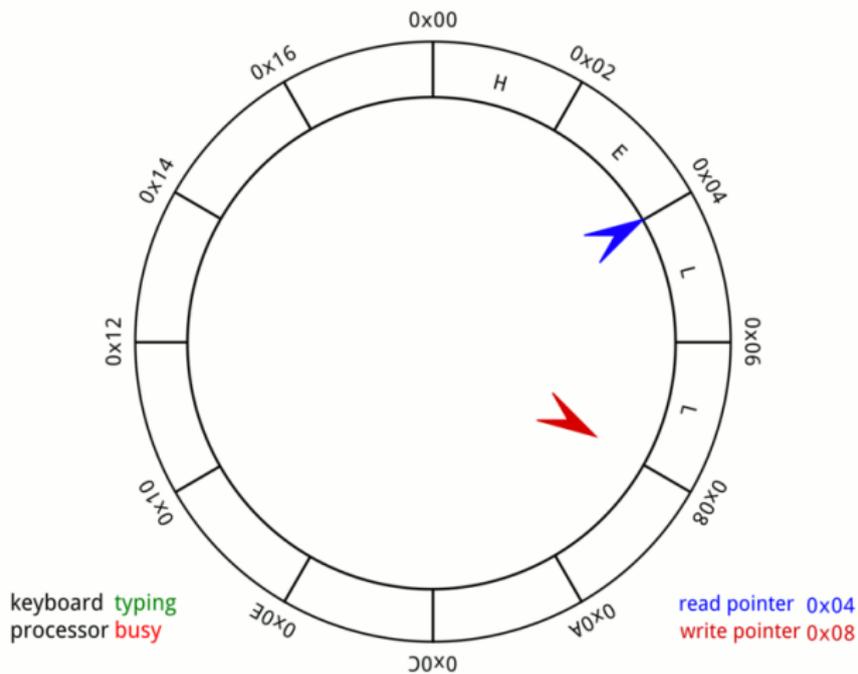
By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

# Coda basata su vettore circolare



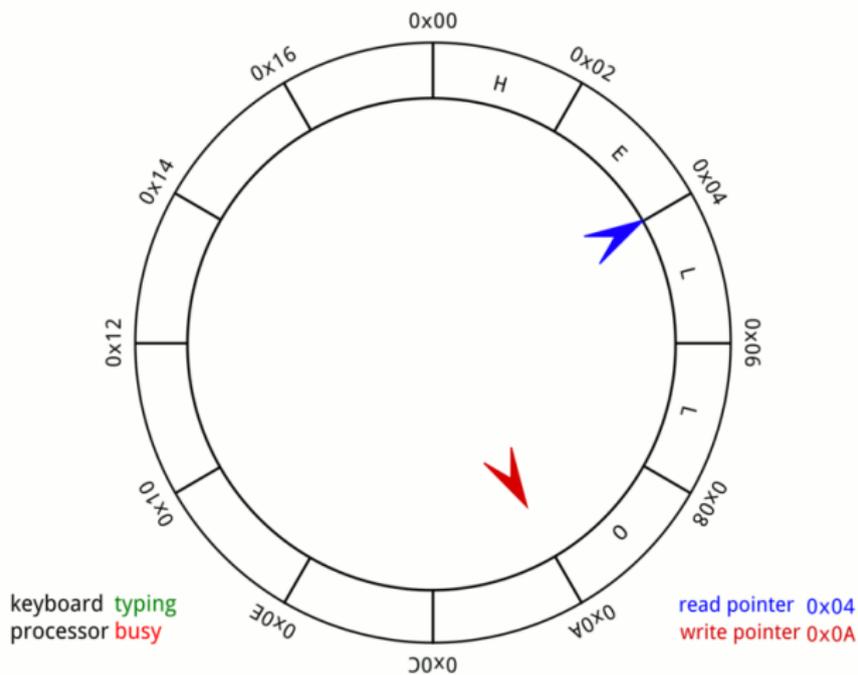
By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

# Coda basata su vettore circolare



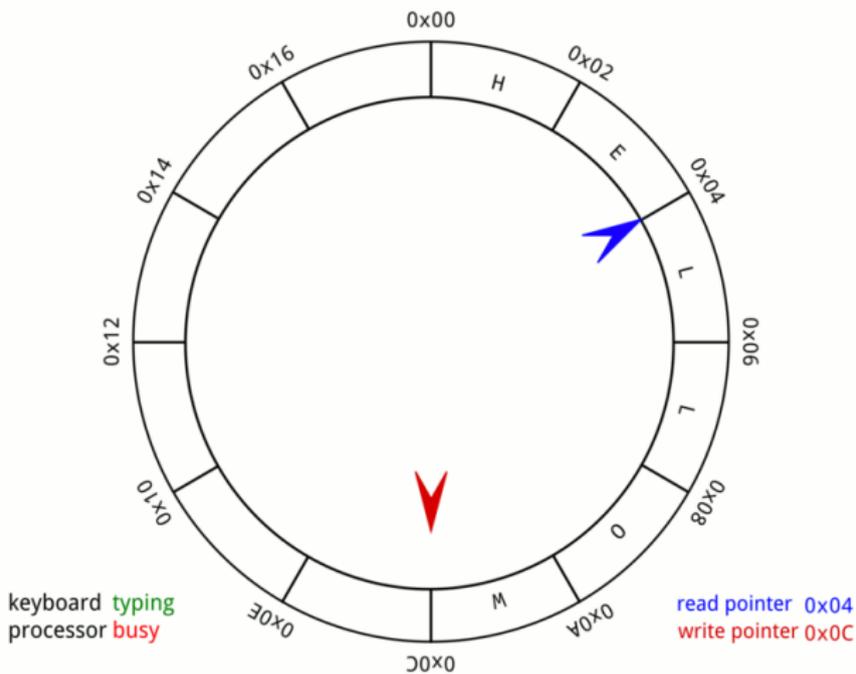
By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

# Coda basata su vettore circolare



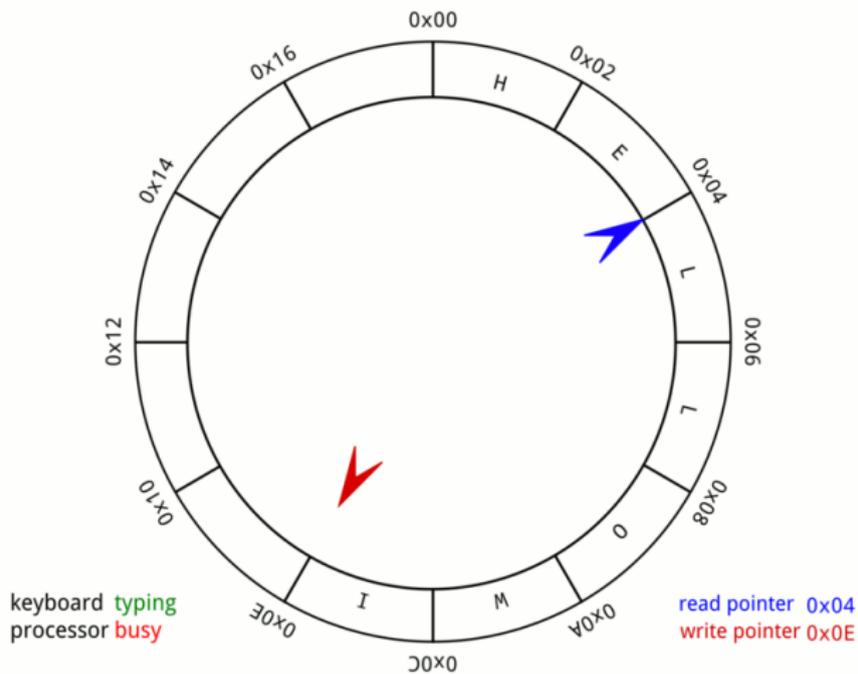
By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

# Coda basata su vettore circolare



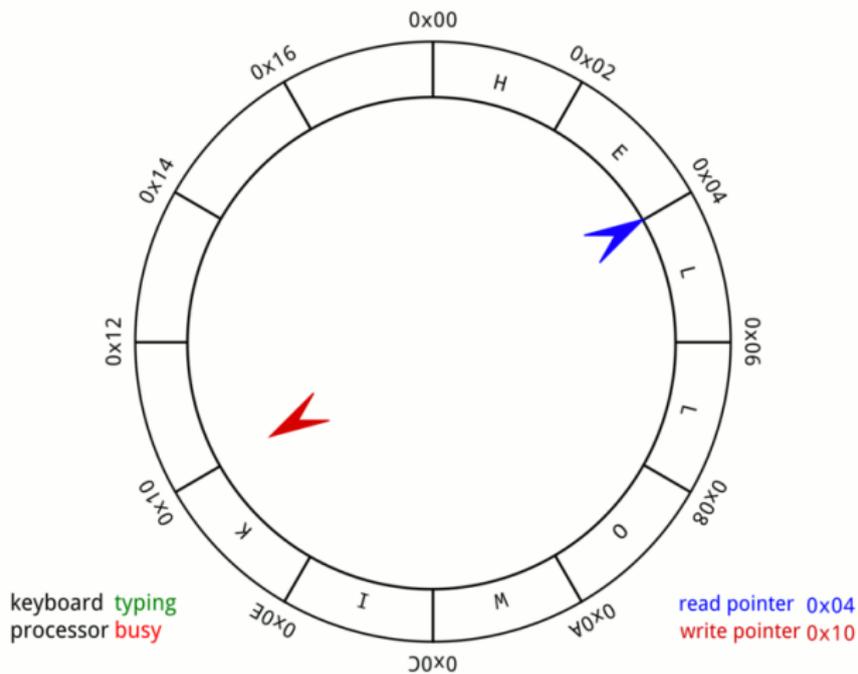
By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

# Coda basata su vettore circolare



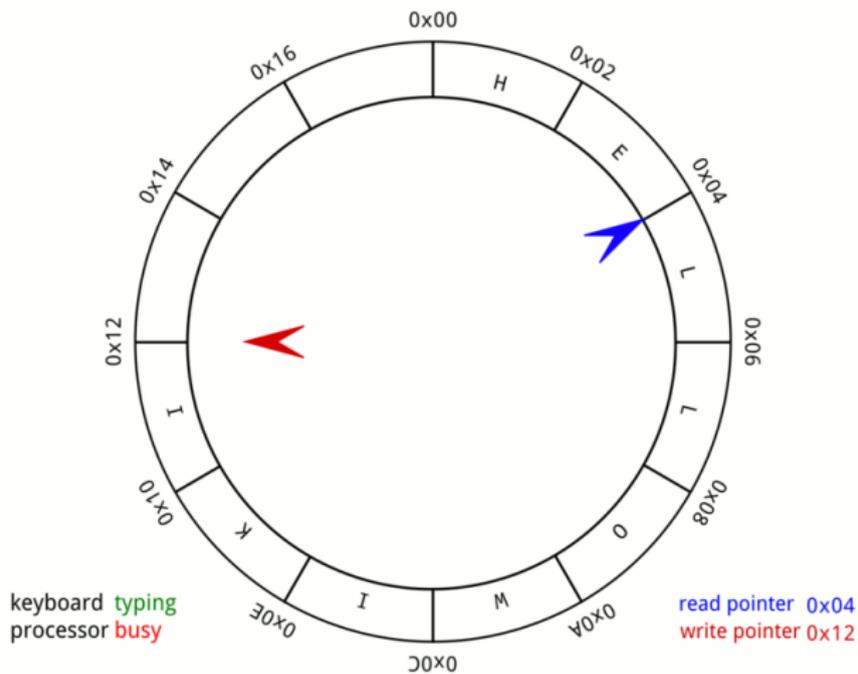
By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

# Coda basata su vettore circolare



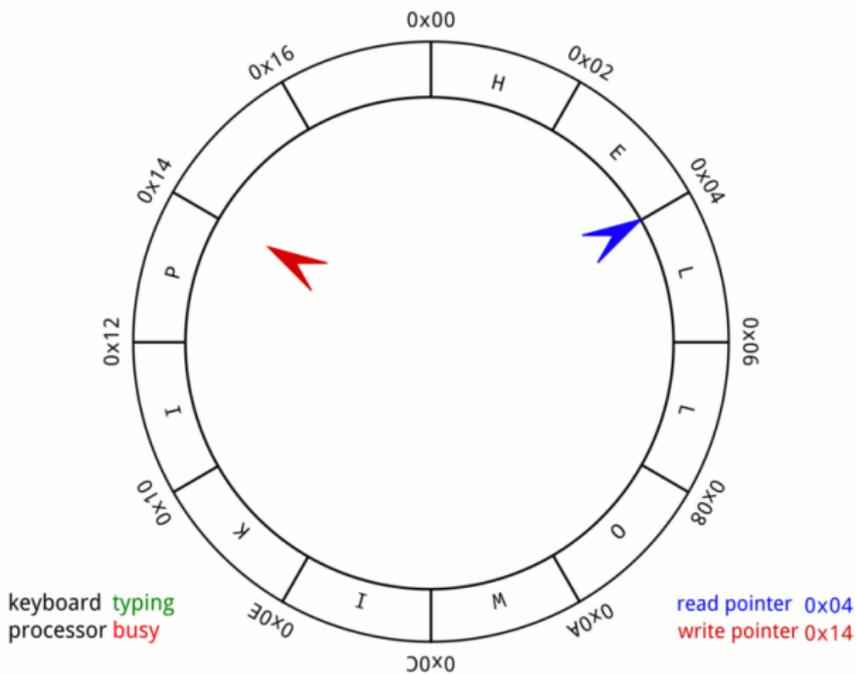
By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

# Coda basata su vettore circolare



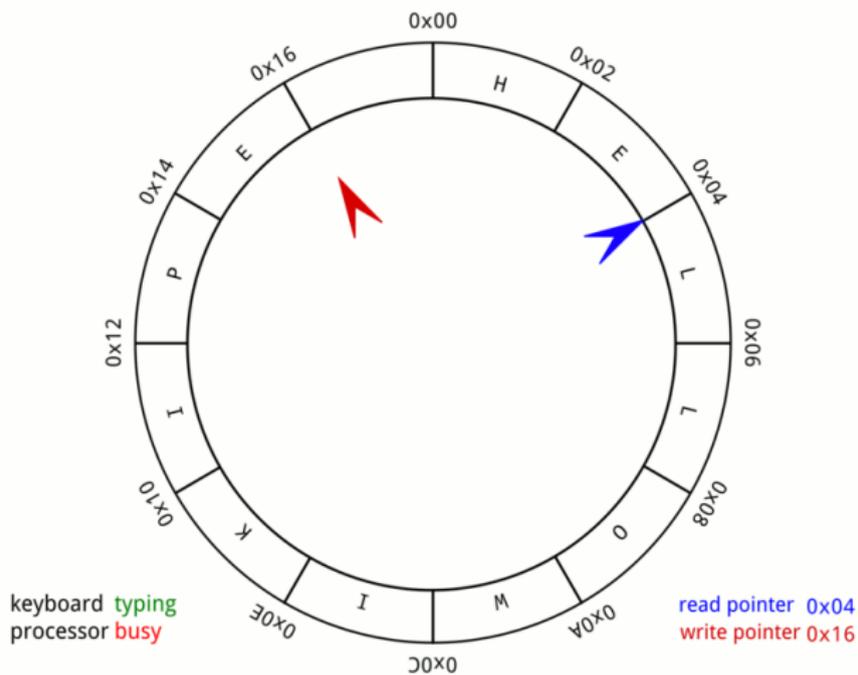
By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

# Coda basata su vettore circolare



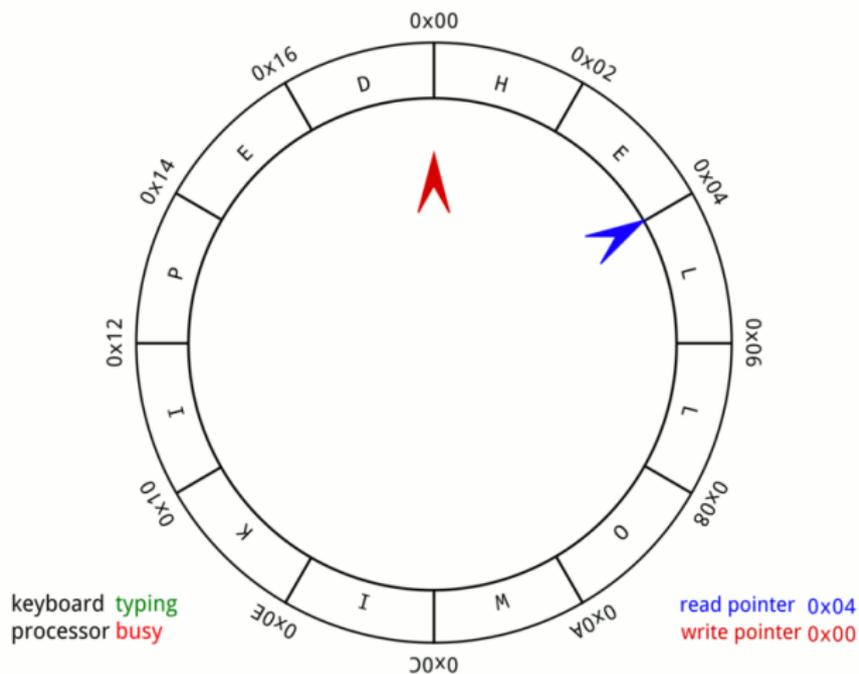
By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

# Coda basata su vettore circolare



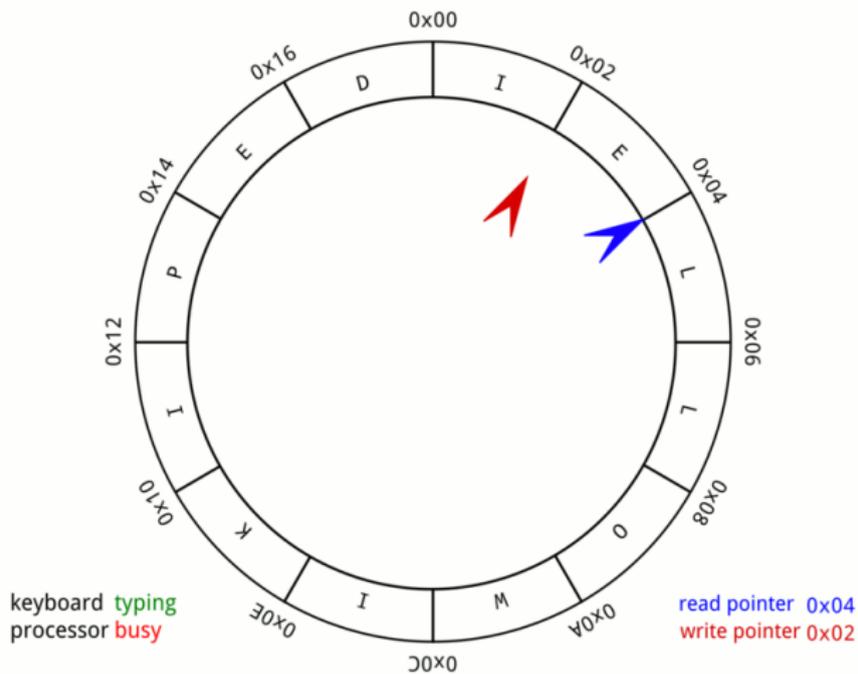
By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

# Coda basata su vettore circolare



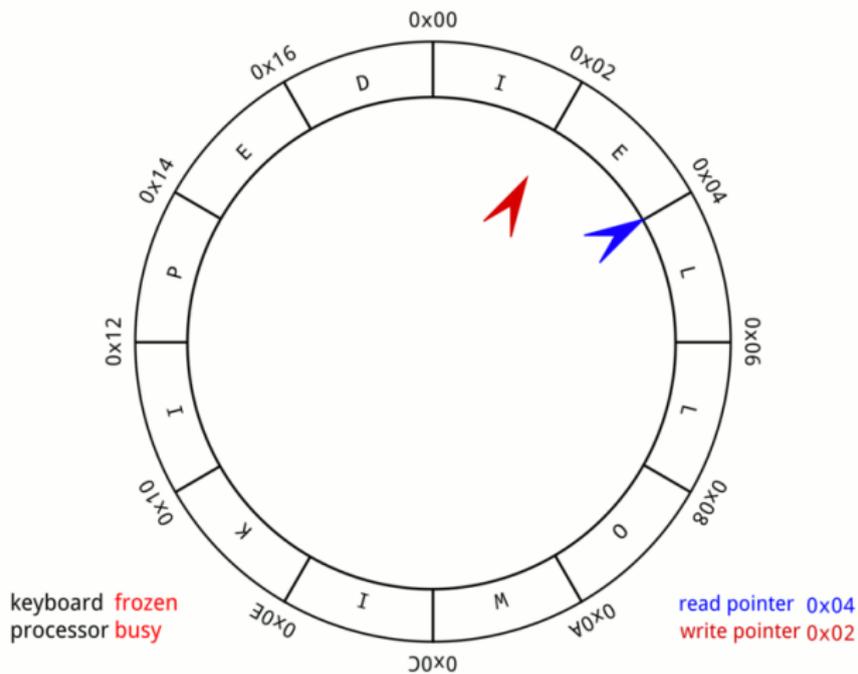
By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

# Coda basata su vettore circolare



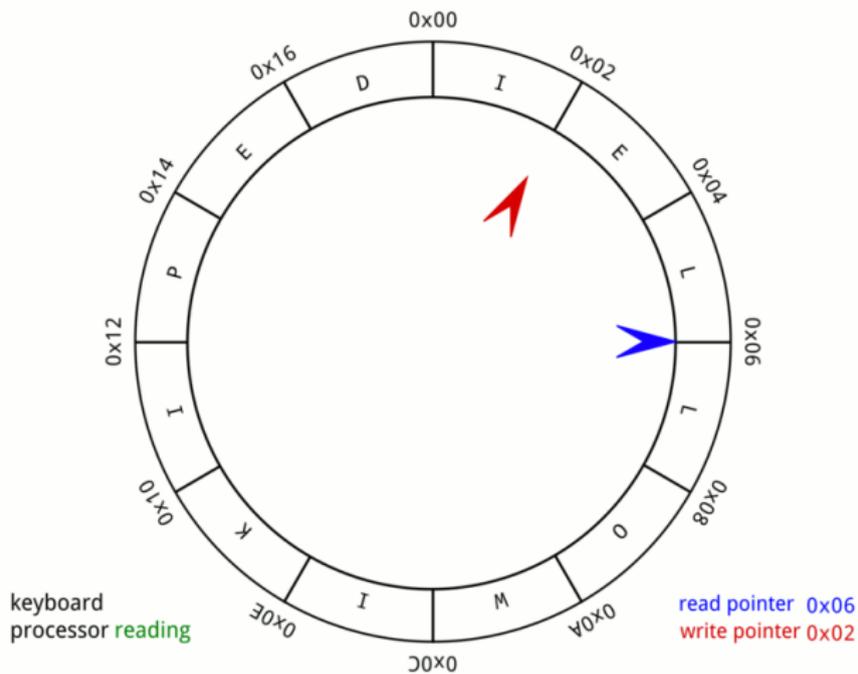
By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

# Coda basata su vettore circolare



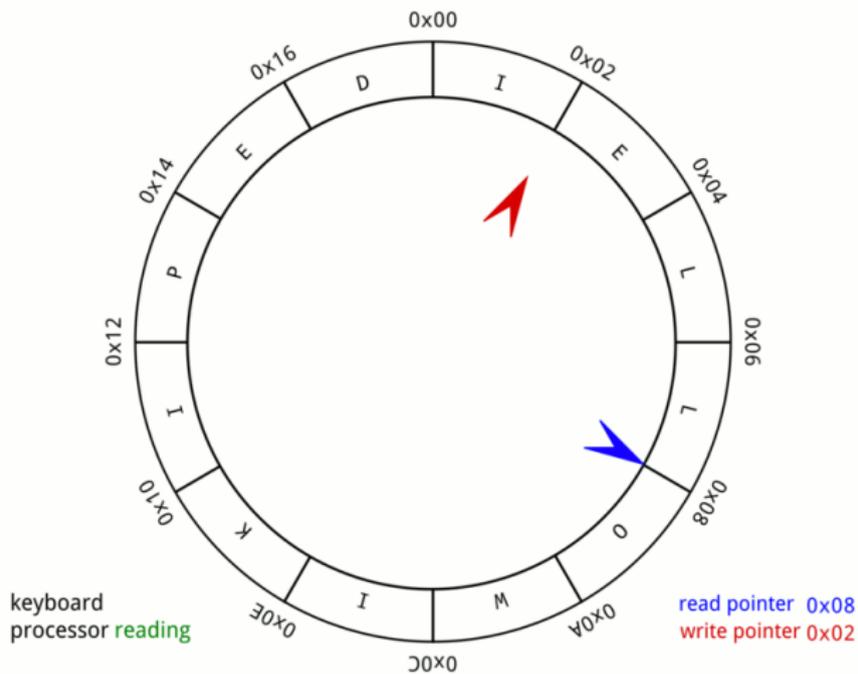
By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

# Coda basata su vettore circolare



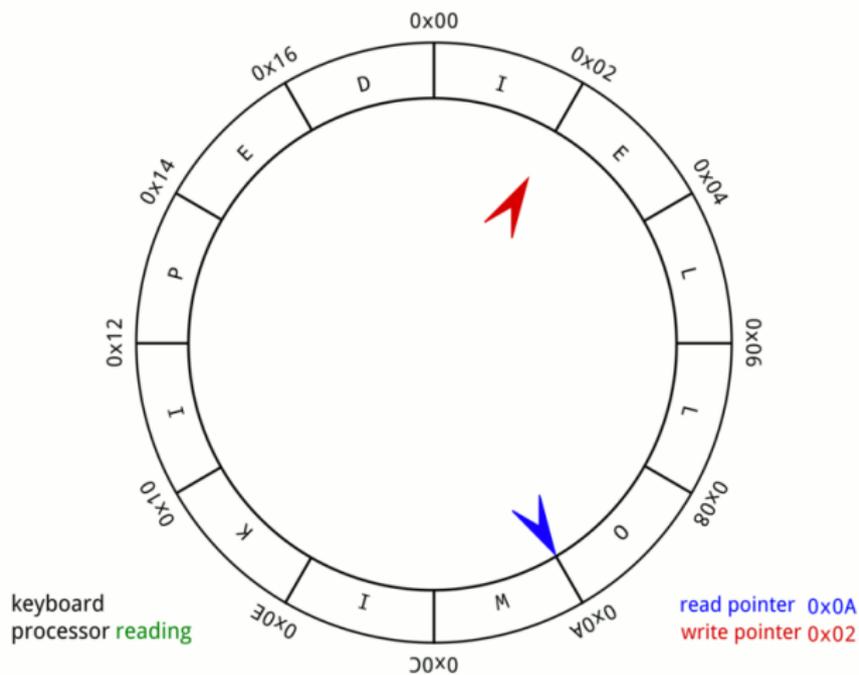
By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

# Coda basata su vettore circolare



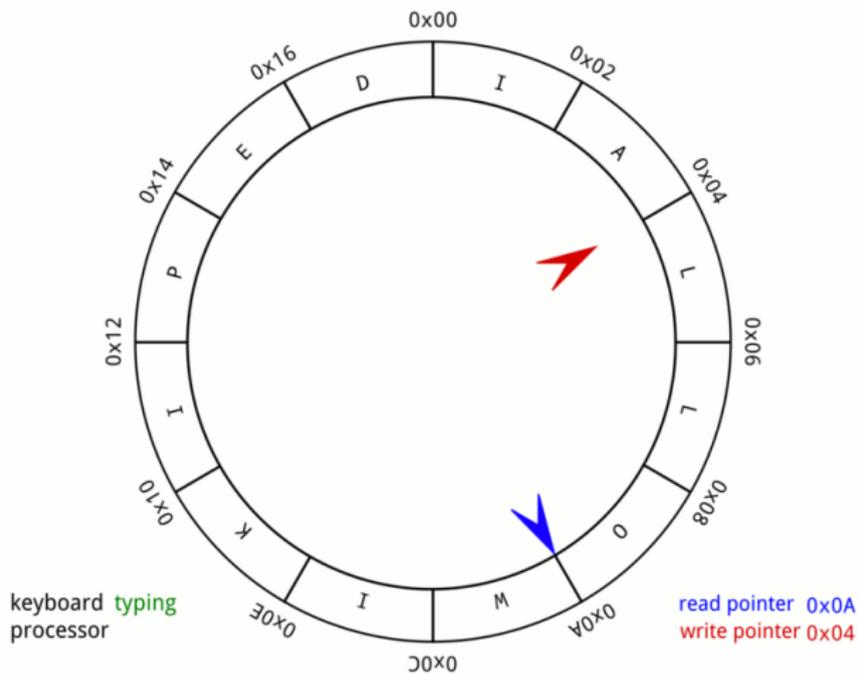
By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

# Coda basata su vettore circolare



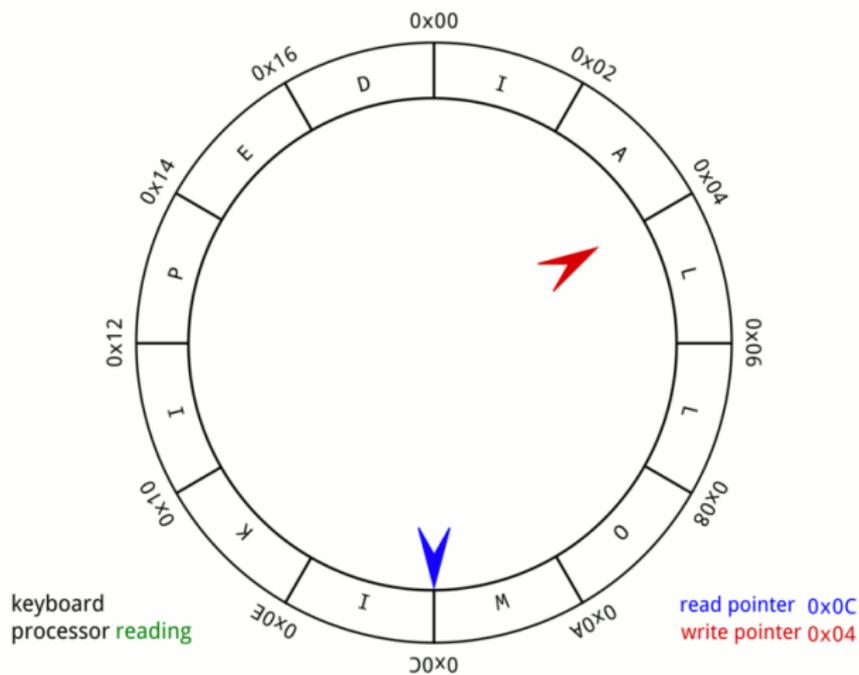
By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

# Coda basata su vettore circolare



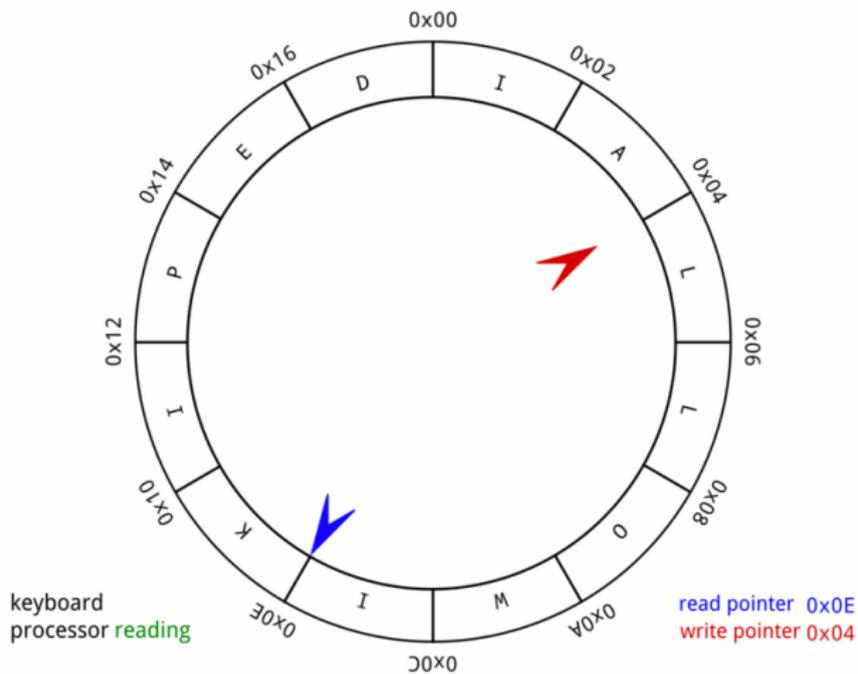
By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

# Coda basata su vettore circolare



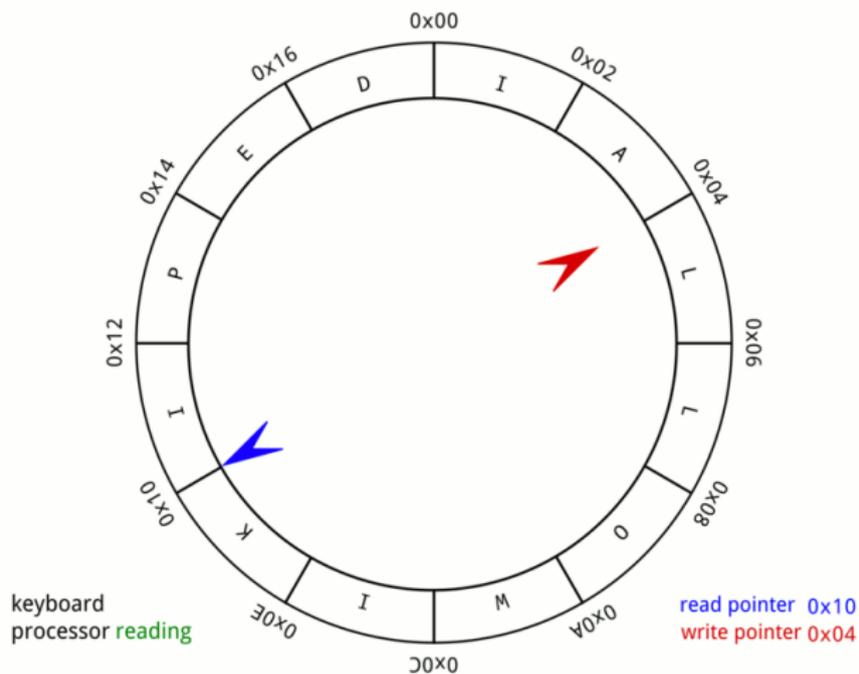
By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

# Coda basata su vettore circolare



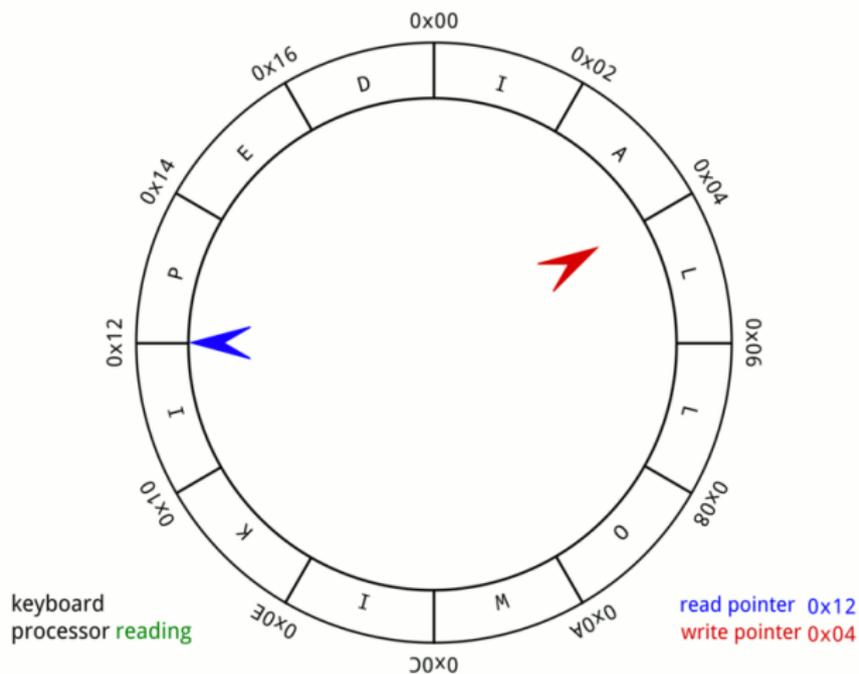
By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

# Coda basata su vettore circolare



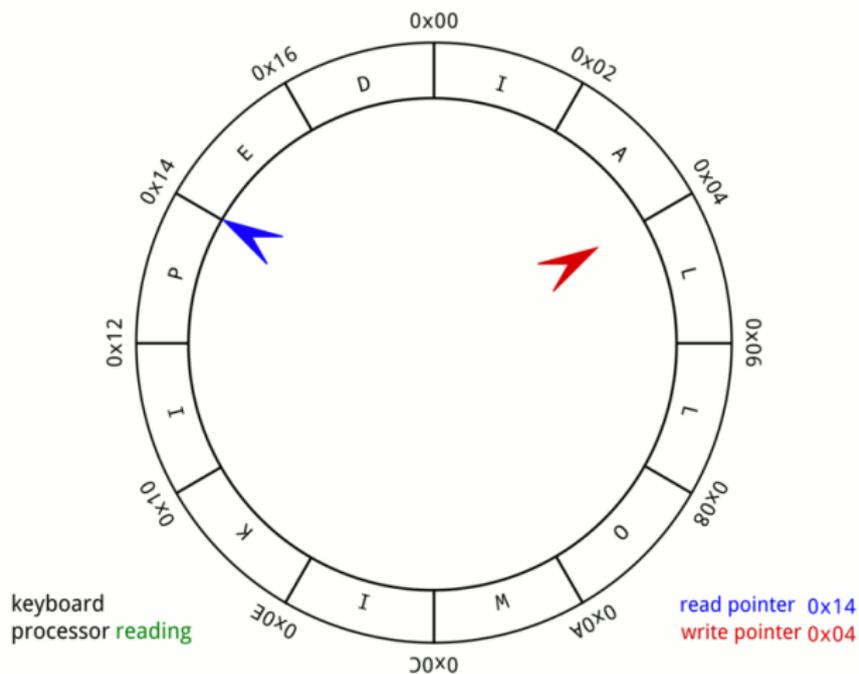
By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

# Coda basata su vettore circolare



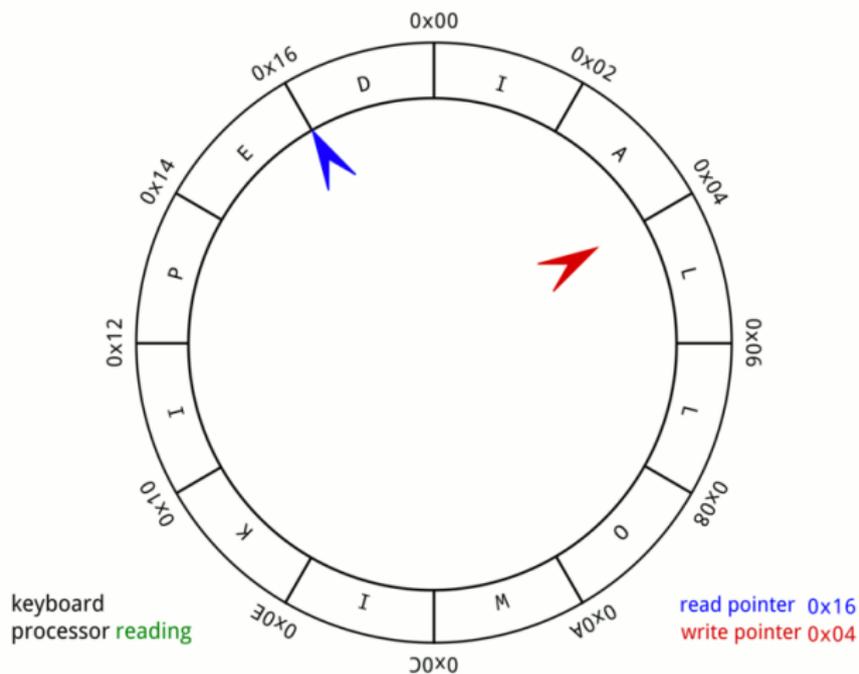
By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

# Coda basata su vettore circolare



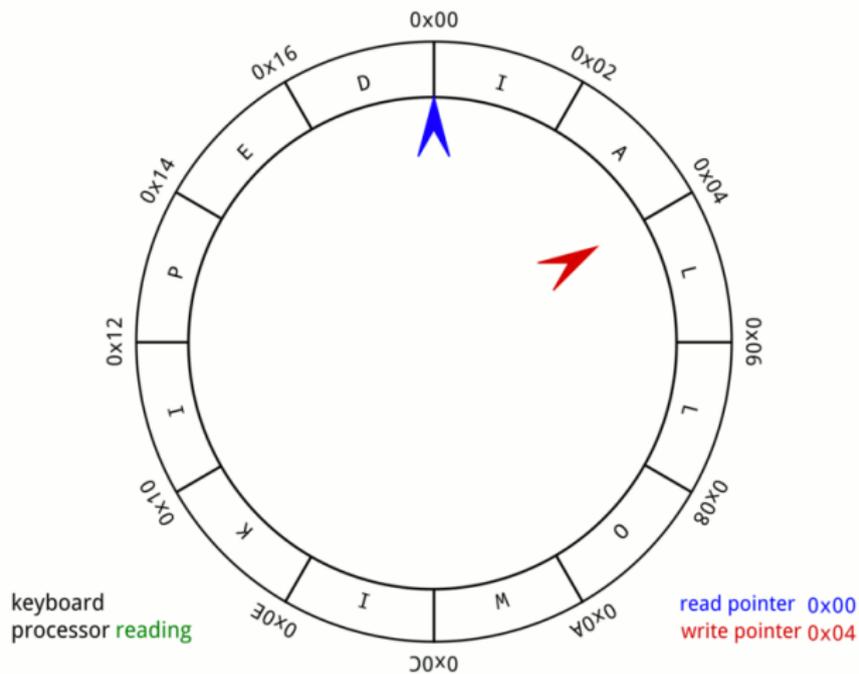
By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

# Coda basata su vettore circolare



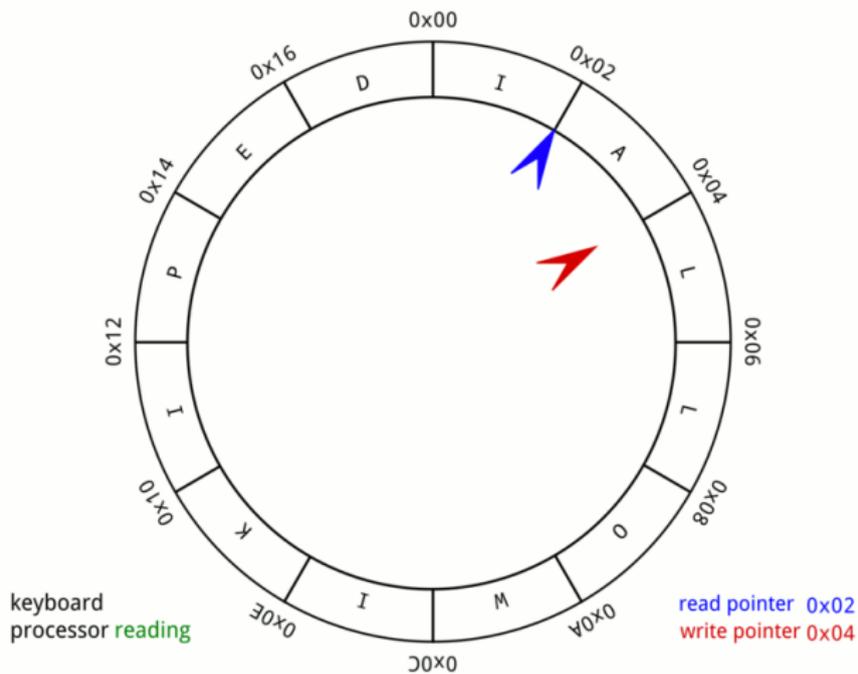
By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

# Coda basata su vettore circolare



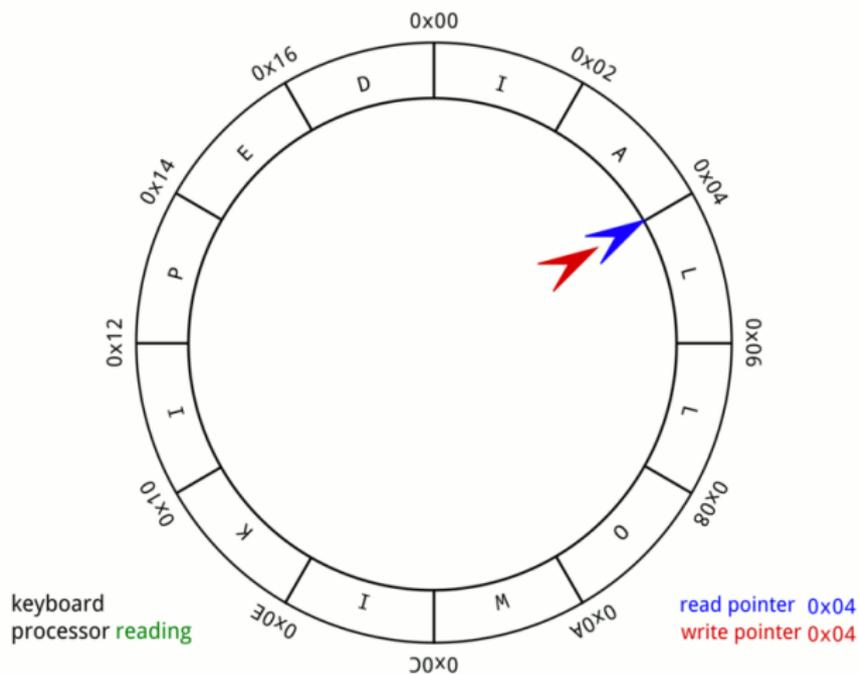
By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

# Coda basata su vettore circolare



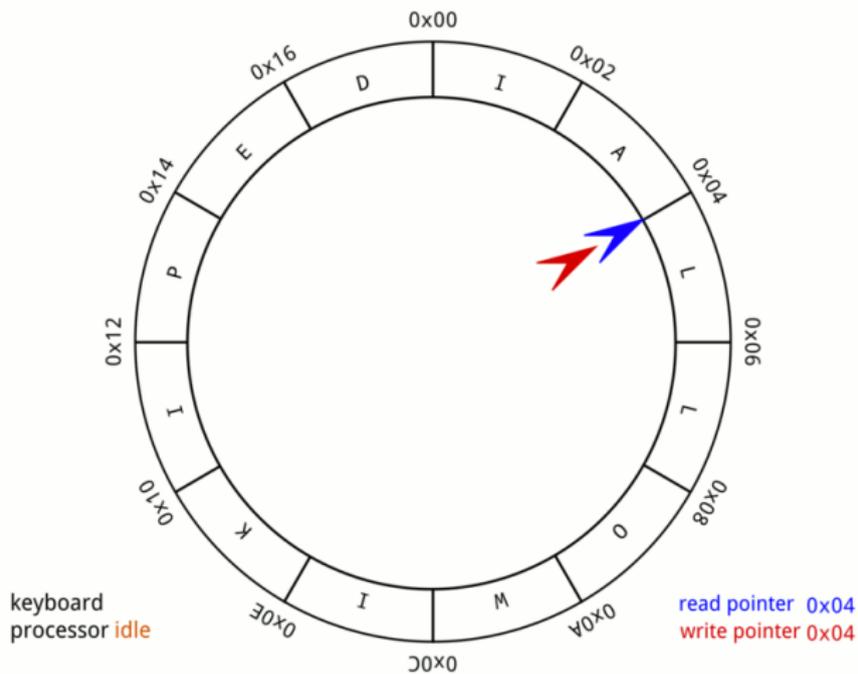
By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

# Coda basata su vettore circolare



By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

# Coda basata su vettore circolare



By MuhannadAjjan [CC BY-SA 4.0] (<http://creativecommons.org/licenses/by-sa/4.0>) via Wikimedia

## Coda basata su vettore circolare – Pseudocodice

## QUEUE

```

ITEM[] A           % Elementi
int n              % Dim. attuale
int testa          % Testa
int m              % Dim. massima

```

```

QUEUE Queue(int dim)

```

```

    QUEUE t = new QUEUE
    t.A = new int[0...dim - 1]
    t.m = dim
    t.testa = 0
    t.n = 0
    return t

```

```

ITEM top()

```

```

    precondition: n > 0
    return A[testa]

```

```

boolean isEmpty()

```

```

    return n == 0

```

```

ITEM dequeue()

```

```

    precondition: n > 0

```

```

    ITEM t = A[testa]
    testa = (testa + 1) mod m
    n = n - 1
    return t

```

```

enqueue(ITEM v)

```

```

    precondition: n < m

```

```

    A[(testa + n) mod m] = v
    n = n + 1

```

# Coda basata su vettore circolare – Java

```
public class VectorQueue implements Queue {  
  
    /** Element vector */  
    private Object[] A;  
  
    /** Current number of elements in the queue */  
    private int n;  
  
    /** Top element of the queue */  
    private int head;  
  
    public VectorQueue(int dim) {  
        n = 0;  
        head = 0;  
        A = new Object[dim];  
    }  
  
    public boolean isEmpty() {  
        return n==0;  
    }  
}
```

## Coda basata su vettore circolare – Java

```
public Object top() {
    if (n == 0)
        throw new IllegalStateException("Queue is empty");
    return A[head];
}

public Object dequeue() {
    if (n == 0)
        throw new IllegalStateException("Queue is empty");
    Object t = A[head];
    head = (head+1) % A.length;
    n = n-1;
    return t;
}

public void enqueue(Object v) {
    if (n == A.length)
        throw new IllegalStateException("Queue is full");
    A[(head+n) % A.length] = v;
    n = n+1;
}
}
```