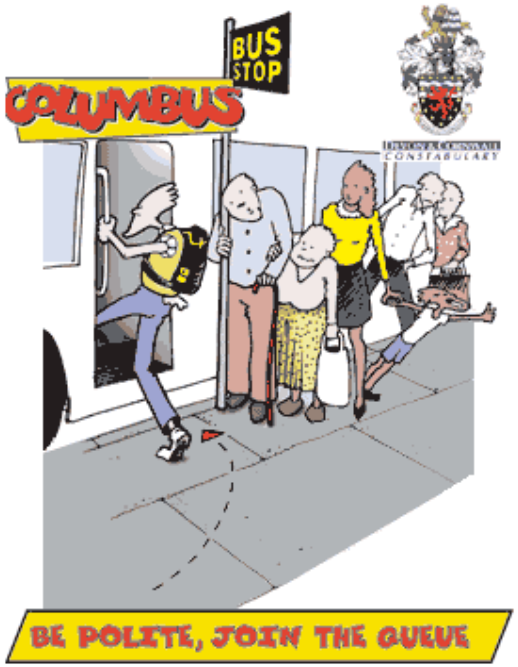
In una coda



gli inserimenti si fanno alla fine e le cancellazioni all'inizio!
First In First Out

CODA: i requisiti

Una coda (queue) è una struttura dati ad accesso regolamentato: gli inserimenti possono avvenire solo alla "fine" (rear) e le cancellazioni possono avvenire solo all' "inizio" (front) della coda

Quindi necessariamento il primo estratto è il primo che era stato inserito.

Non si può estrarre (dequeue) un elemento da una coda vuota, nè si può inserire (enqueue) un elemento in una pila piena.

CODA: la specificazione

```
/* La specificazione della coda*/


typedef struct coda * CodaP;

CodaP  costrCoda( int max_num);
/* alloca la memoria per una nuova coda che può contenere fino a max_num
elementi.
 Prec: max_num >0
 postc: restituisce un puntatore alla nuova coda, NULL se non c'è memoria */

void distrCoda(CodaP q);
/*prec:  q è una coda valida
postc: libera la memoria impegnata dalla coda */

void  azzera(CodaP q);
/* azzera una coda consentendo di riutilizzarla senza riallocare la memoria
senza modificare q -> numMax.
 Prec: q è una coda valida
 postc: restituisce la coda svuotata*/
```

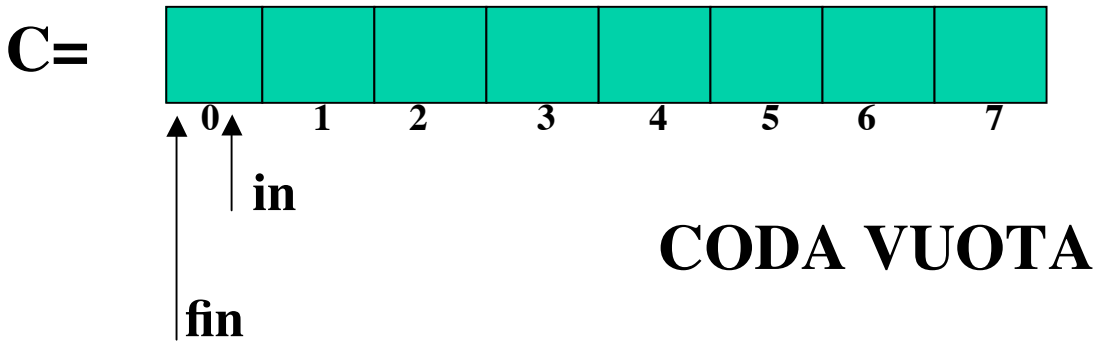
CODA: la specificazione

```
int  vuota(const CodaP q);
/* da' vero se la coda e' vuota, falso altrimenti
*prec: q è una coda valida
postc: restituisce un valore !=0 se q è vuota, 0 altrimenti*/

int  piena(const CodaP q);
/* da' vero se la coda e' piena, falso altrimenti
*prec: q è una coda valida
postc: restituisce un valore !=0 se q è piena, 0 altrimenti*/

int    primo(const CodaP q);
/* legge e restituisce il primo elemento in coda, se non e' vuoto
*prec: q è una coda valida && !vuota(q)
postc: restituisce il primo elemento in coda */
```

CODA: la specificazione

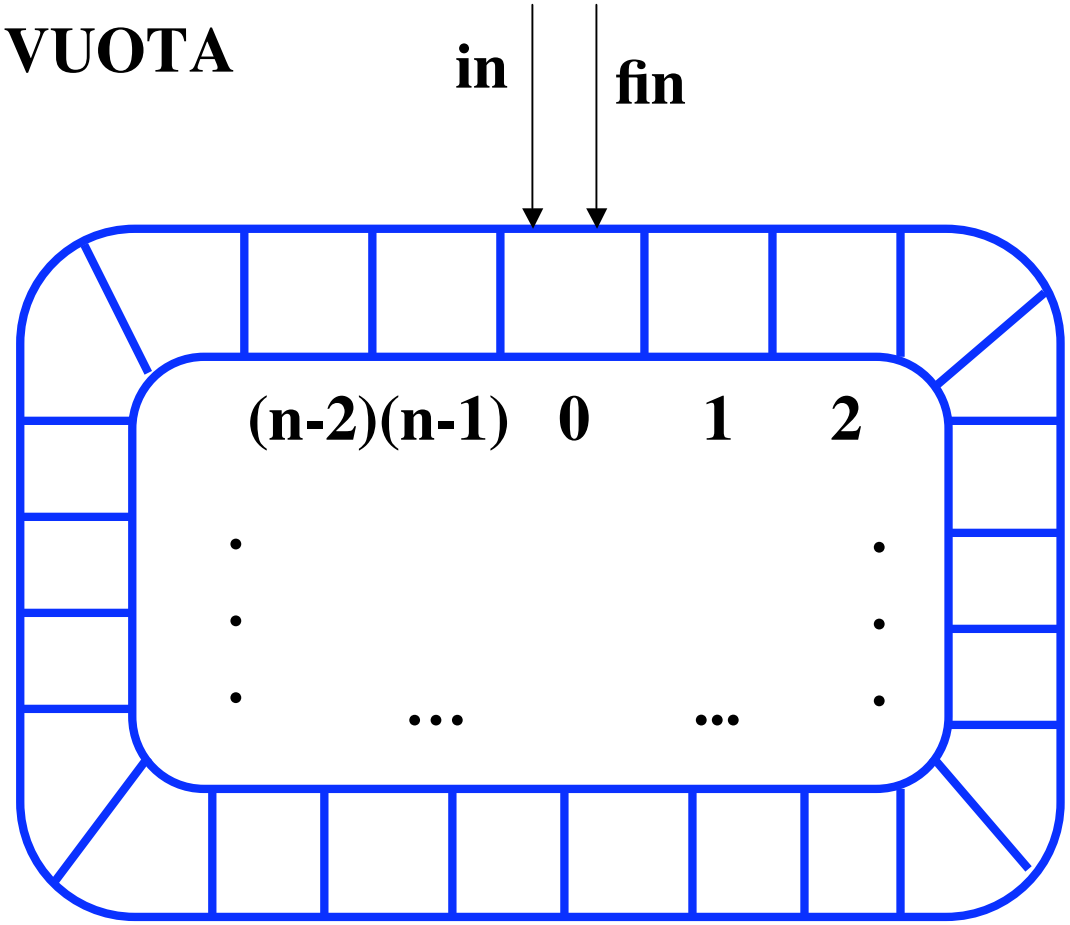
```
void  accoda(int el, CodaP q);
/* accoda el nella coda
prec: (q è una coda valida  && !piena(q))
postc: el è accodato in q, se c'è abbastanza memoria, esce altrimenti*/

int  estrae(CodaP q);
/* restituisce ed elimina il primo elemento inserito nella coda, se non e' vuota
*prec: q è una coda valida &&  !vuota(q)
postc: restituisce il  primo elemento, eliminandolo dalla coda  */
```

CODA: implementazione su un vettore, versione 1



Gli elementi della coda sono quelli nel vettore C compresi tra C[in] e C[fin-1] inclusi.
L'inserimento avviene in C[fin] e poi fin è incrementato di uno.

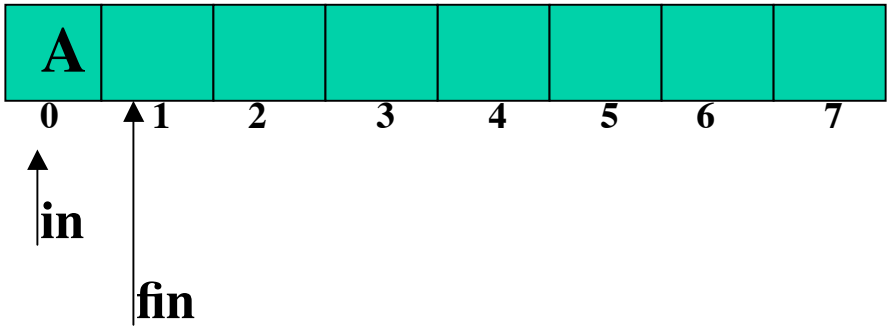
CODA VUOTA



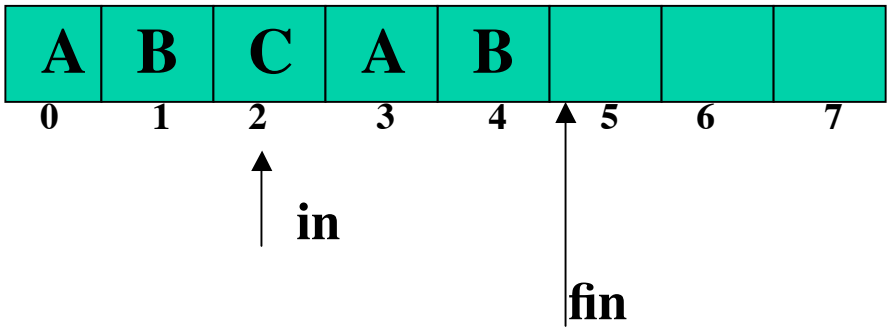
L'incremento di fin dopo l'inserimento deve avvenire modulo n, la dimensione del vettore.

CODA: implementazione su un vettore, versione 1

passo 1: un inserimento



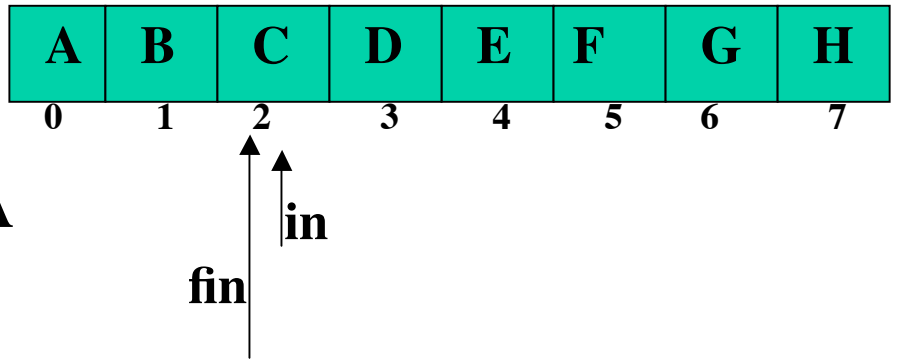
dopo qualche inserimento ed estrazione:



L'estrazione avviene salvando C[in] e incrementando in, sempre modulo la dimensione della coda

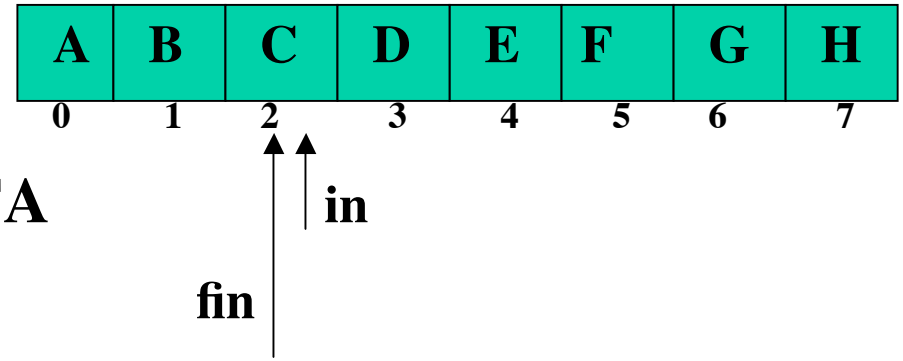
CODA: implementazione su un vettore, versione 1

continuando ad inserire...:



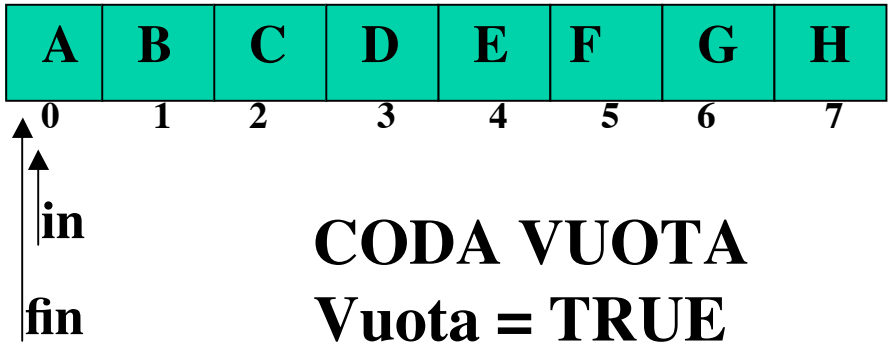
in = fin <=> CODA PIENA

continuando ad estrarre...:



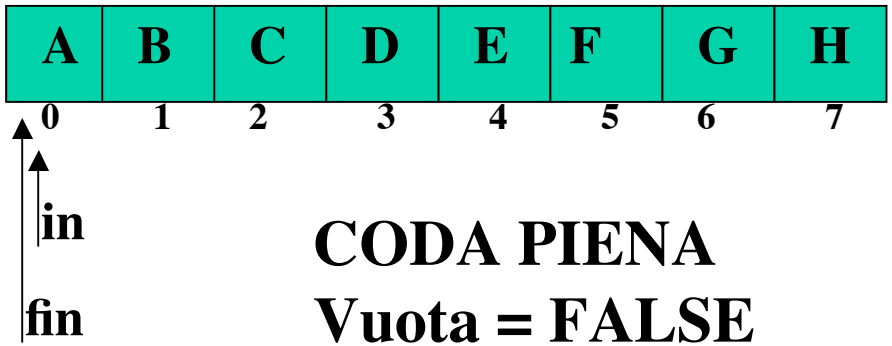
in = fin <=> CODA VUOTA

CODA: implementazione su un vettore, versione 1



CODA VUOTA
Vuota = TRUE

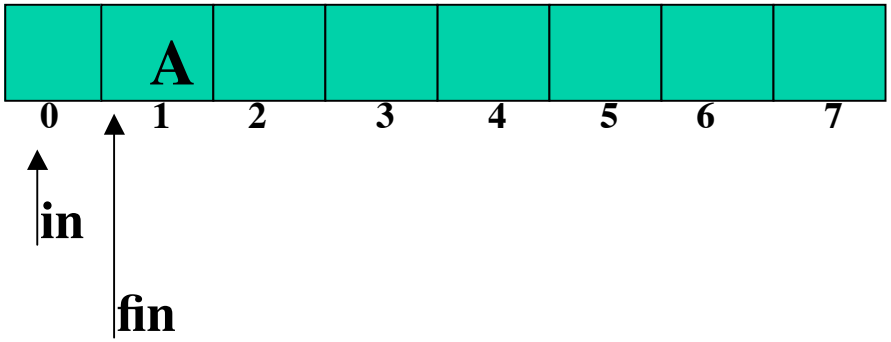
Per distinguere i due casi introduciamo una variabile booleana, Vuota, inizialmente posta a TRUE e che ad ogni inserimento diventa FALSE



CODA PIENA
Vuota = FALSE

ad ogni estrazione, che può avvenire solo se Vuota è FALSE, si controlla se l'incremento di in porterà allo stesso valore di fin e in tal caso Vuota diventa di nuovo TRUE.

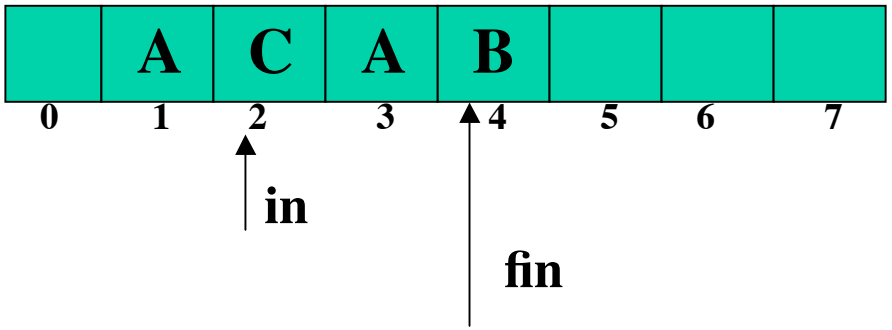
CODA: implementazione su un vettore, versione 2



Sacrificando una entrata del vettore si evita di dover introdurre una nuova variabile.
Gli elementi della coda sono quelli nel vettore C compresi tra C[in+1] e C[fin] inclusi.
L'inserimento avviene incrementando fin, modulo la dimensione del vettore, e poi copiando in C[fin] il valore voluto.

CODA: implementazione su un vettore, versione 2

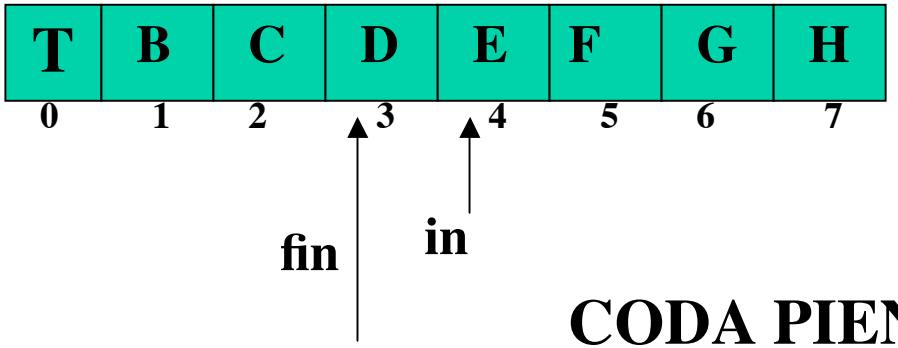
Dopo l'estrazione di A
primo = A



L'estrazione, come nella versione 1, avviene salvando l'elemento C[in] e poi incrementando in, modulo la dimensione del vettore

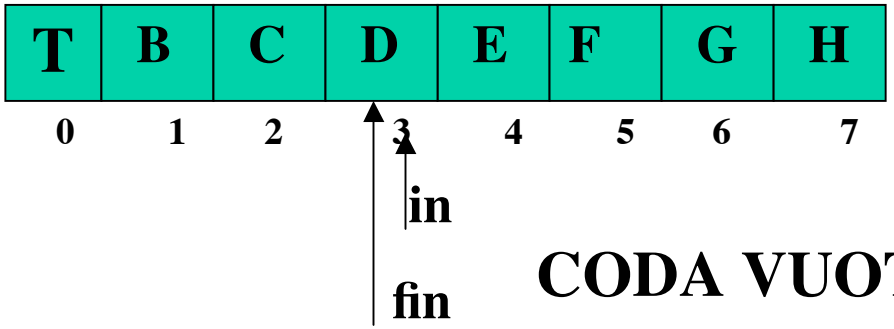
CODA: implementazione su un vettore, versione 2

Continuando a inserire, si avra' che fin+1 (modulo la dimensione del vettore) è uguale a in. Ma allora la coda è piena!



CODA PIENA!!

Continuando a estrarre, si avra' che in (modulo la dimensione del vettore) è uguale a fin. Ma allora la coda è vuota!



CODA VUOTA!!

CODA: implementazione su un vettore, versione 2. Il codice.

```
/* Nome file : codaVett.c
Implementazione della coda su vettore, gli elementi della coda  sono compresi
* tra c->inizio+1 e c->fine  se c è una variabile di tipo  struct coda, con
0  ≤ c->inizio,c->fine ≤ FULL */

#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "Coda.h"


struct coda {
 int inizio;
 int fine;
 int numMax;
 int * elCoda;
};
```

CODA: implementazione su un vettore, versione 2. Il codice.

```
CodaP  costrCoda( int numMaxEl)
/* alloca la memoria per una nuova coda che può contenere fino a max_num
elementi.
 Prec: numMaxEl >0
 postc: restituisce un puntatore alla nuova coda, NULL se non c'è memoria */
{CodaP q;
assert(numMaxEl >0);
q = (CodaP) calloc(1,sizeof(struct coda));/*inizio=fine = numMax=0 el Coda = NULL*/
if (!q) printf("non c'è memoria per costCoda");
q ->numMax = numMaxEl;
q->elCoda = calloc(numMaxEl,sizeof(int));
if (!q->elCoda) {q=NULL;printf("non c'è memoria per costrCoda");}
return q;}

void distrCoda(CodaP q)
/*prec:  q != NULL
postc: libera la memoria impegnata dalla pila */
{assert(q);
free(q -> elCoda);
free(q);
}
```

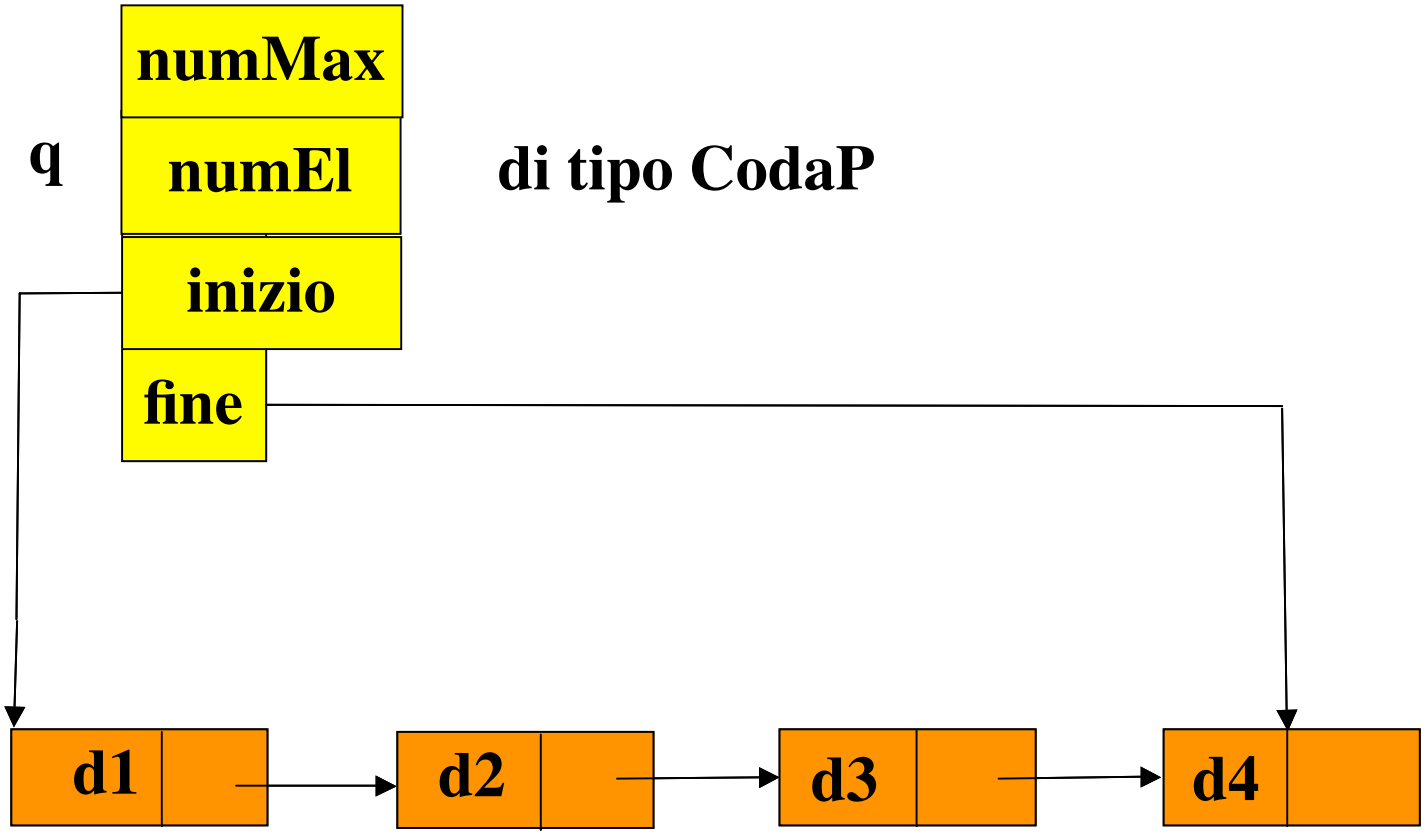
CODA: implementazione su un vettore, versione 2. Il codice.

```
void  azzera(CodaP q)
/* azzera una coda consentendo di riutilizzarla senza riallocare la memoria e
senza modificare q -> numMax.
 Prec: q != NULL
 postc: restituisce la coda svuotata*/
{assert(q);
q -> inizio = 0;
q -> fine = 0;}

int vuota(const CodaP q)
/* da' vero se la coda e' vuota, falso altrimenti
*prec:q != NULL
postc: restituisce unvalore !=0 se q è vuota, 0 altrimenti*/
 { assert(q);
   return (q->inizio == q->fine);}

int piena(const CodaP q)
/* da' vero se la coda e' piena, falso altrimenti
*prec: q != NULL
postc: restituisce unvalore !=0 se q è piena, 0 altrimenti*/
{assert(q);
 return (q->inizio == (q->fine + 1) % q->numMax);}
```

CODA: implementazione su un vettore, versione 2. Il codice.

```
int primo( CodaP q)
/* legge e restituisce il primo elemento in coda, se non e' vuoto
*prec: q è una coda valida && !vuota(q)
postc: restituisce il primo elemento in coda */
{assert(q);
return q->elCoda[(q->inizio + 1) % q->numMax];}

void  accoda(int el, CodaP q)
/* accoda el nella coda
prec: (q != NULL && !piena(q))
postc: el è accodato in q*/
{assert(!piena(q));
q->fine = (q->fine + 1) % q->numMax;
 q->elCoda[q->fine ] = el;}


int     estrae(CodaP q)
/* restituisce ed elimina il primo elemento inserito nella coda, se non e' vuota
*prec: q != NULL &&  !vuota(q)
postc: restituisce il  primo elemento, eliminandolo dalla coda  */
{assert(!vuota(q));
q->inizio = (q->inizio + 1) % q->numMax;
return q->elCoda[q->inizio];
}
```

CODA: implementazione su una lista.



CODA: implementazione su una lista. Il codice

```
/* Nome file: codaLista.c
coda implementata con una lista */

#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "Coda.h"

struct elem {              /* un elemento della coda */
  int  dato;
  struct elem   *next;
};

typedef   struct elem*   Elem;

struct coda {
   int    numEl;
   int    numMax;              /* un contatore per gli elementi*/
   Elem   inizio;            /* un puntatore al primo elemento*/
   Elem   fine;             /* un puntatore all'ultimo elemento*/
};
```

CODA: implementazione su una lista. Il codice

```
CodaP  costrCoda( int numMaxEl)
/* alloca la memoria per una nuova coda che può contenere fino a max_num elementi.
 Prec: numMaxEl >0
 postc: restituisce un puntatore alla nuova coda, NULL se non c'è memoria */
{CodaP q;
assert(numMaxEl >0);
q = (CodaP) calloc(1,sizeof(struct coda));/*inizio=fine = NULL, numMax=numEl=0 */
if (!q) printf("non c'è memoria per costCoda");
q ->numMax = numMaxEl;
return q;}
```

CODA: implementazione su una lista. Il codice

```
void azzera(CodaP q)
/* azzera una coda consentendo di riutilizzarla senza riallocare la memoria
senza modificare q -> numMax.
 Prec: q != NULL
 postc: restituisce la coda svuotata*/
{ Elem temp;
 assert(q != NULL);
   q -> numEl = 0;
   while (q->inizio)
              {temp = q->inizio;
              q -> inizio = q-> inizio-> next;
              free(temp);
              }
 q -> fine = NULL;
 }
```

CODA: implementazione su una lista. Il codice

```
void distrCoda(CodaP q)
/*prec:  q != NULL
postc: libera la memoria impegnata dalla pila */
{ Elem temp;
assert(q);
while (q->inizio)
            {temp = q->inizio;
            q -> inizio = q-> inizio-> next;
            free(temp);
            }
free(q);
}
```

CODA: implementazione su una lista. Il codice

```
int vuota( CodaP q)
/* da' vero se la coda e' vuota, falso altrimenti
*prec:q != NULL
postc: restituisce un valore !=0 se q è vuota, 0 altrimenti*/
{assert(q != NULL);
return ( (q -> numEl == 0));}

int piena( CodaP q)
/* da' vero se la coda e' piena, falso altrimenti
*prec: q != NULL
postc: restituisce un valore !=0 se q è piena, 0 altrimenti*/
{assert(q != NULL);
return  q -> numEl == q -> numMax;}

int primo( CodaP q)
/* legge e restituisce il primo elemento in coda, se non e' vuoto
*prec: q != NULL && !vuota(q)
postc: restituisce il primo elemento in coda */
{assert(!vuota(q));
return (q -> inizio -> dato);}
```

CODA: implementazione su una lista. Il codice

```
int estrae(CodaP q)
/* restituisce ed elimina il primo elemento inserito nella coda, se non e' vuota
*prec: q != NULL &&  !vuota(q)
postc: restituisce il  primo elemento, eliminandolo dalla coda  */
{ int   d;
  Elem   p;
  assert(!vuota(q));
  d = q -> inizio -> dato;
  p = q -> inizio;
  q -> inizio = q -> inizio -> next;
  q -> numEl--;
  free(p);
  return d;}
```

CODA: implementazione su una lista. Il codice

```
void accoda(int el, CodaP q)
/* accoda el nella coda
prec: (q!= NULL  && !piena(q))
postc: el è accodato in q, se non c'è memoria per el esce*/
{ Elem   p;
 assert(!piena(q));
 p = (Elem) calloc(1,sizeof(struct elem));
 if (!p) {printf("non c'è memoria per accoda");exit(1);}
 p -> dato = el;
 if (!vuota(q))
              {q -> fine -> next = p;
               q -> fine = p;}
 else
               q -> fine = q -> inizio = p;
 q -> numEl++;
}
```

CODA: esempio di uso. Il codice

```
/*Nome del file: scheduler.c
Si usano due code per gestire nell'ordine di arrivo le richieste di servizi dirette a
due processori*/

#include <stdio.h>
#include <assert.h>
#include "Coda.h"

#define MAX 100

int main(void)
{int max = MAX;
  int     c, cnt_a = 0, cnt_b = 0;        /*serviranno per numerare le richieste accodate*;
  unsigned int    pid;                        /* e' il numero identificativo del processo che
                                              richiede il servizio */

  CodaP   a, b;
  a = costrCoda(max);
  b = costrCoda(max);
```

CODA: esempio di uso. Il codice

```
/*Fase di accodamento delle richieste */
  printf("\n scrivi A o B a seconda che sia una richiesta per ilprocessore A o B,\
          C per terminare\n");
  while ((c = getchar()) != 'C')
         {switch (c)
              {case 'A':
                  printf("\n scrivi un numero che rappresenta una richiesta per il processore A\n");
                  assert(scanf("%u", &pid) == 1); /* inserisci i numeri  identificativi delle richieste */
                  if (!piena(a))
                  accoda(pid, a);
                   printf("\n scrivi A o B a seconda che sia una richiesta per ilprocessore A o B,\
                                  C per terminare\n");
                  break;
             case 'B':
                 printf("\n scrivi un numero che rappresenta una richiesta per il processore A\n");
                                  C per terminare\n");
                assert(scanf("%u", &pid) == 1); /* inserisci i numeri  identificativi delle richieste */
                if (!piena(b))
                accoda(pid, b);
                printf("\n scrivi A o B a seconda che sia una richiesta per ilprocessore A o B,\
                                  C per terminare\n");
                break;
          }
      }
```

CODA: esempio di uso. Il codice

```
/*  Preleva dalla coda e stampa le richieste */
   printf(" il primo della coda a è %u\n",primo(a));
   printf(" il primo della coda b è %u\n",primo(b));
   printf("%s","\nElenco richieste processore A:\n");
   while (!vuota(a))
        { pid = estrae(a);
          printf("La richesta %u è %d\n", ++cnt_a, pid);
         }
   putchar('\n');
   printf("%s","\nElenco richieste processore B:\n");
   while (!vuota(b))
       { pid = estrae(b);
         printf("La richiesta %u è %d\n", ++cnt_b, pid);
       }
   putchar('\n');
   distrCoda(a);
   distrCoda(b);
   return 0;
}
```

CODA: esempio di uso. Una delle due implementazioni può essere utilizzata senza alcuna modifica!