UNIVERSITY OF TRENTO - Italy
Department of Information
and Communication Technology

Laboratorio 19

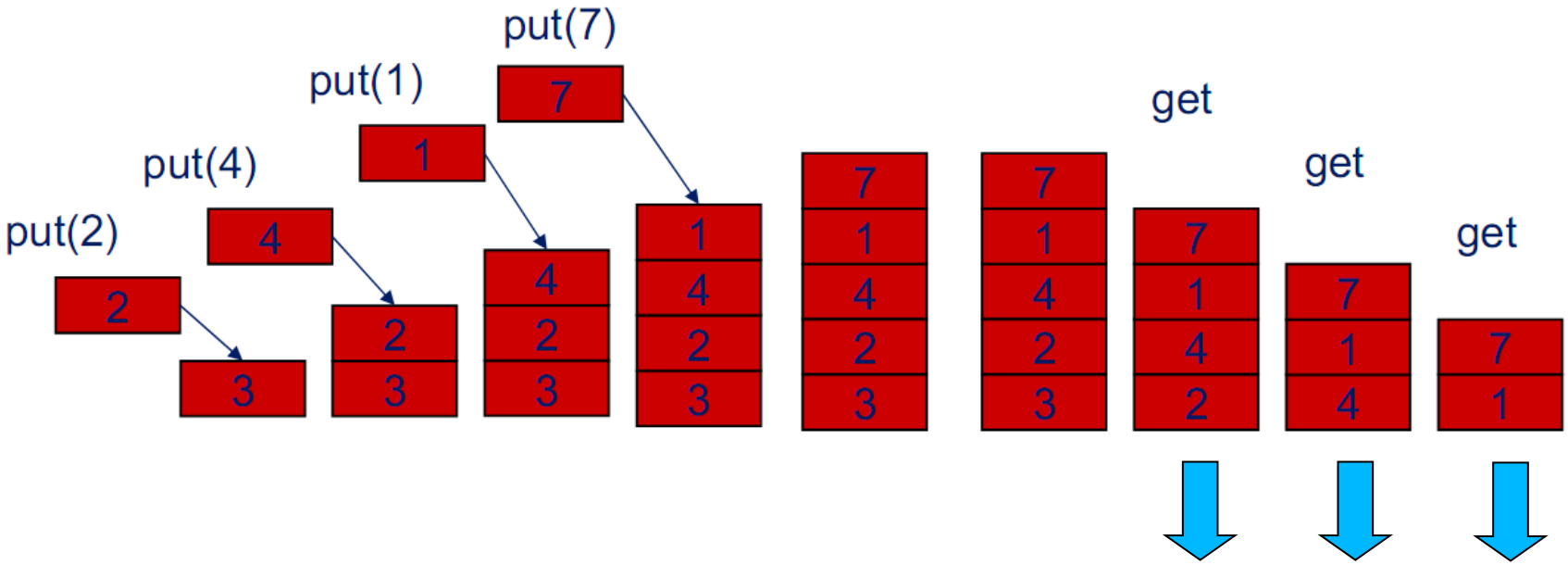
DISI – aa 2024/25

Pierluigi Roberti
Carmelo Ferrante

Università degli Studi di Trento

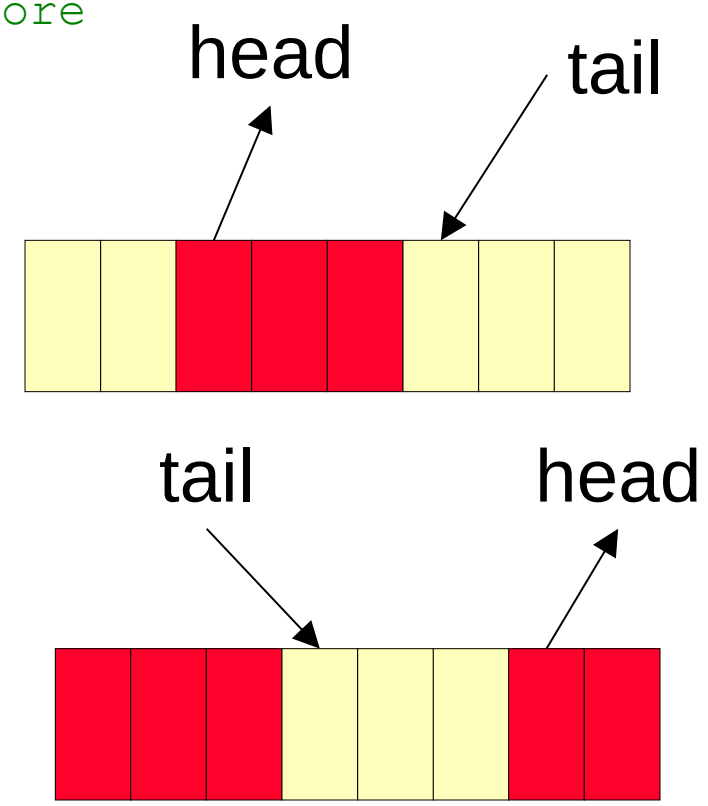
ADT: Coda

- Uno **coda,** o lista **FIFO** (first in first out, come in una coda in biglietteria) è una struttura dati astratta (ADT) che consente due operazioni:
  - Inserimento di un nuovo elemento in fondo alla coda (**put**)
  - Estrazione di un elemento dalla testa della coda (**get**)



Coda (FIFO) con array

```
typedef struct TipoCoda {
   int n;          // numero elementi nella coda
   int dim;        // dimensione max coda
   int head;       // posizione elemento in testa
   int tail;       // posizione elemento in coda
   Tdato *s;        // vettore elementi
   TipoCoda (int x) { // costruttore
      dim = x;
      n = 0;
      head = 0;
      tail = 0;
      s = new Tdato[x];
   }
  // costruttore default
  // distruttore
  // metodo stampa
} TipoCoda;
typedef TipoCoda Coda;
typedef TipoCoda* CodaPtr;
```



!! Attenzione !!

- Implementazione diversa da quella vista a lezione

- Lezione

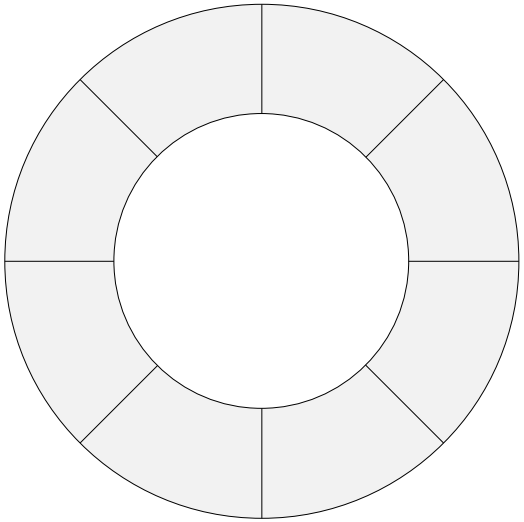
```
typedef struct TipoCoda {
  int dim;// dimensione max coda
  int head;// posizione testa
  int tail; // posizione coda
  Tdato *s;      // vettore elementi
  TipoCoda (int x) {
    dim = x;
    head = 0;
    tail = 0;
    s = new Tdato[x];
  }
} TipoCoda;
```

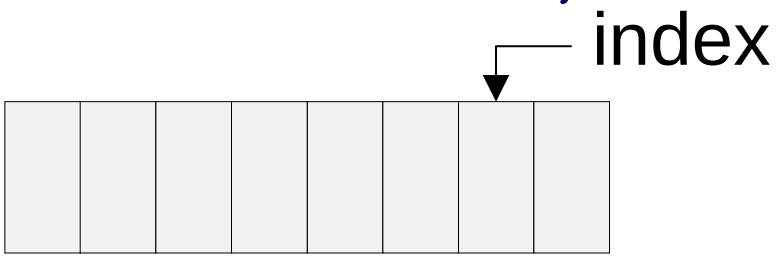
- Esercitazione

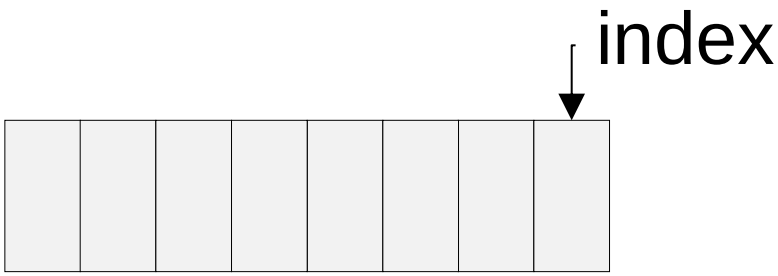
```
typedef struct TipoCoda {
   int n; // numero elementi in coda
   int dim;// dimensione max coda
   int head; // posizione testa
   int tail; // posizione coda
   Tdato *s; // vettore elementi
   TipoCoda (int x) {
      dim = x;
      n = 0;
      head = 0;
      tail = 0;
      s = new Tdato[x];
   }
} TipoCoda;
```

Coda (FIFO) con array

- Implementazione con array
  - Array circolare
  - Dimensione array: dim

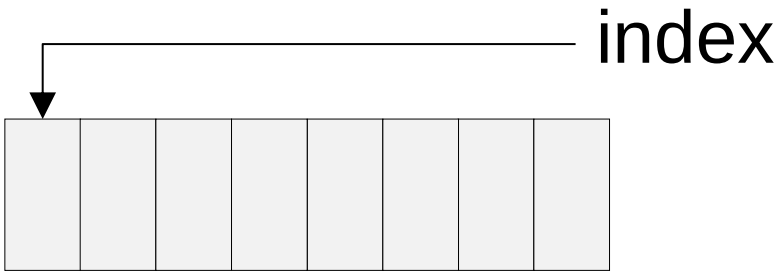






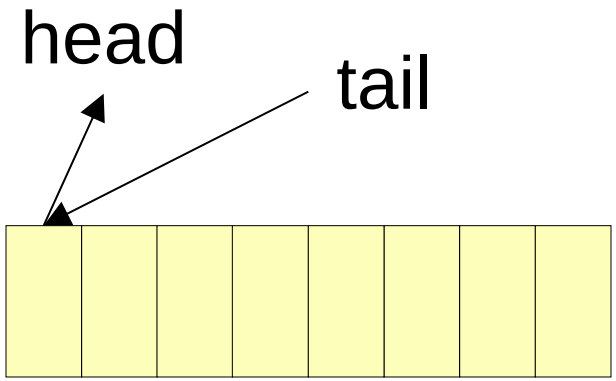
index = (index+1)%dim

oppure



index = ++index%dim

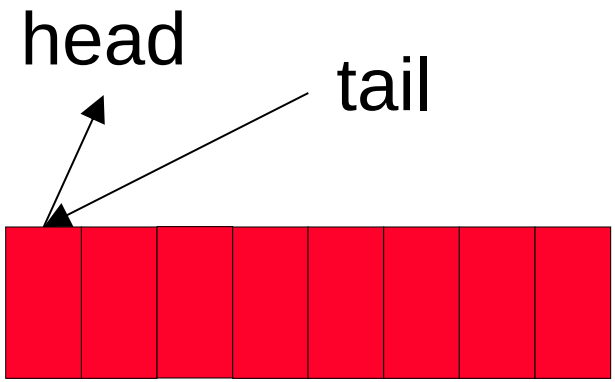
Coda (FIFO) con array



Coda vuota

head = tail ➔ 0

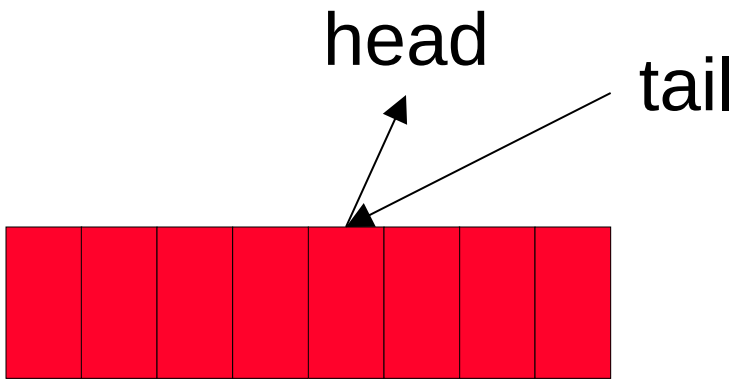
n ➔ 0



Coda piena

head = tail ➔ 0

n ➔ dim



Coda piena

head = tail ➔ k

n ➔ dim

!! Attenzione !!

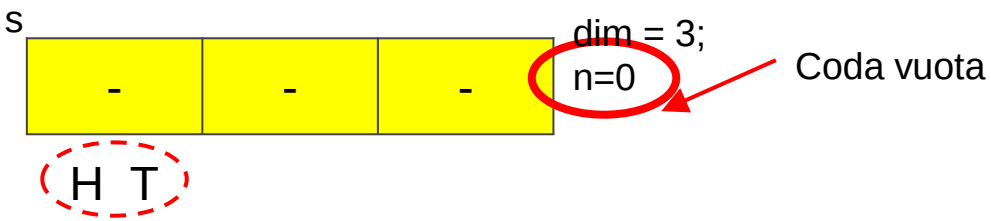
- Implementazione diversa da quella vista a lezione
- Lezione

```
bool codaIsEmpty(TipoCoda* p) {
  return head == tail;
}

bool codaIsFull(TipoCoda* p) {
  return head == ((tail+1)%dim);
}
```

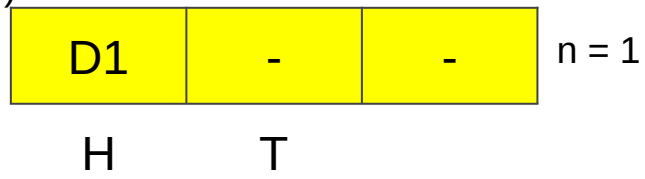
- Esercitazione

```
bool codaIsEmpty(TipoCoda* p) {
  return n == 0;
}

bool codaIsFull(TipoCoda* p) {
  return n == dim;
};
```

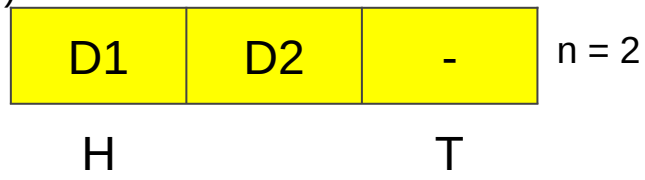
Coda (FIFO) – Esempio



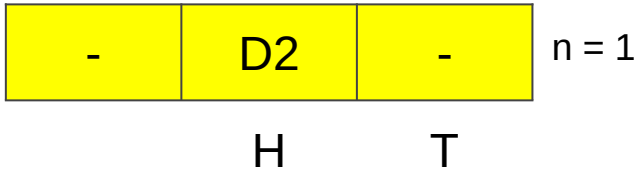
put(D1)



put(D2)

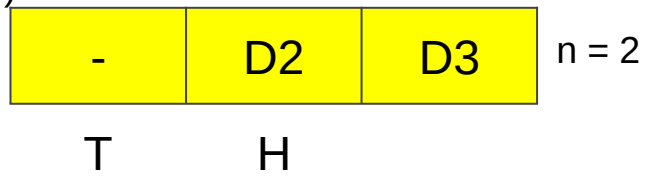


get()

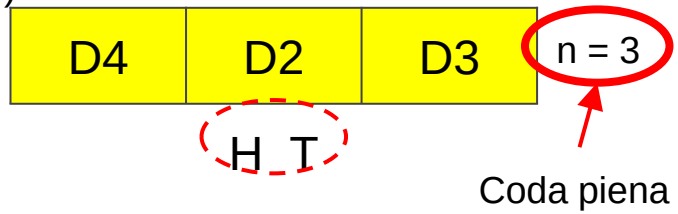


H: head
T: tail

put(D3)



put(D4)

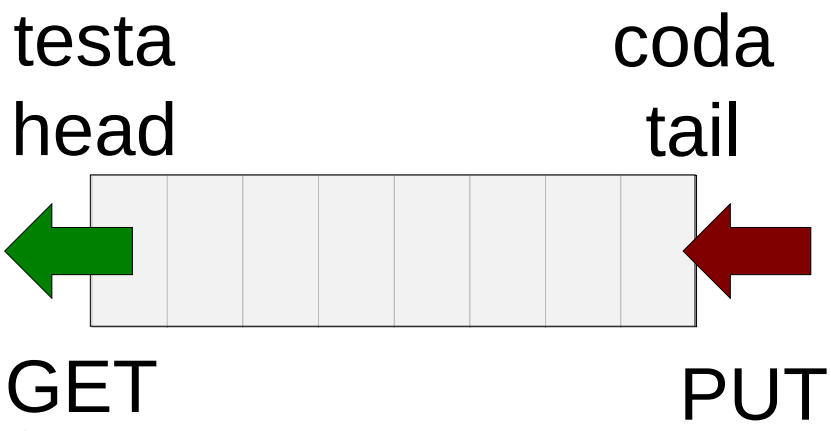


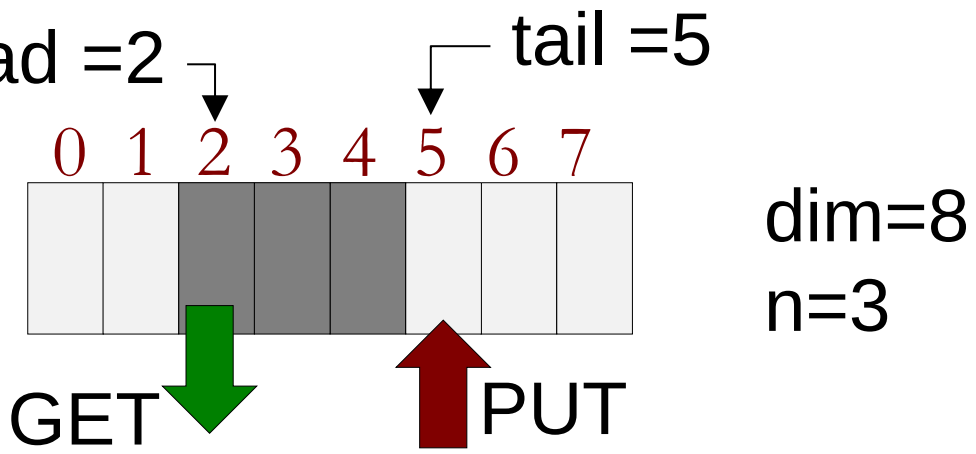
tail = (tail+1) % dim
tail = ++tail % dim

head = (head+1) % dim
head = ++head % dim

Coda (FIFO) con array

- Coda vuota
  - **`n == 0`**
- Coda piena
  - **`n == dim`**
- Coda con elementi
  - PUT: inserimento in coda (tail)
    - **`tail`** → prima posizione libera
  - GET: prelievo in testa (head)
    - **`head`** → posizione elemento da estrarre





Coda FIFO – costruttori

```
// costruttore 0 parametri
TipoCoda () {
        dim = 5;
        n = 0;
        head = 0;
        tail = 0;
        s = new Tdato[5];
}

// costruttore specifico
TipoCoda (int x) {
        dim = x;
        n = 0;
        head = 0;
        tail = 0;
        s = new Tdato[x];
}
```

Coda FIFO – distruttore

```
// distruttore
~TipoCoda () {
        delete[] s;
        // delete s;
}
```

Coda FIFO – metodo stampa v1

```
void print()const{
  if( n==0){
    cout << "coda vuota" << endl;
    return;
  }

  int i = head;
  do {
    s[i].print();
    i = ++i % dim;
  } while( i != tail );

}
```

Funzione di stampa che richiama il metodo di stampa

```
void print(CodaPtr p){
      p->print();
}
```

Coda FIFO – metodo stampa v2

```
void print()const{
  if( n==0){
    cout << “coda vuota” << endl;
    return;
  }

  int i = head;
  s[i].print();
  i = ++i % dim;
  while( i != tail ) {
    s[i].print();
    i = ++i % dim;
  }

}
```

Coda FIFO – metodo stampa v3

```
void print()const{
  if( n==0){
    cout << “coda vuota” << endl;
    return;
  }

  int i=head;
  s[head].print();

} for( i+=1 ; i!=tail ; i=++i%dim )
    s[i].print();
```

Coda FIFO – metodi put e get

```
// prelievo in head
Tdato get(CodaPtr p){
    Tdato d;
    d = p->s[p->head];
    p->n--;
    p->head++; //array va da 0 a dim-1
    p->head = p->head % p->dim;
    return d;
}

// inserimento in tail
void put(CodaPtr p, Tdato d){
   p->n++;
   p->s[p->tail]=d;
   p->tail++; //array va da 0 a dim-1
   p->tail = p->tail % p->dim;
}
```

Esercizio 1 - Code con array

- Vogliamo scrivere un programma che implementa una **coda** di dati interi usando i vettori
- Considerare
  - un TipoCoda come quello descritto in precedenza
  - alias: TipoCoda → Coda e TipoCoda* → CodaPtr
  - il seguente tipo di dato (dichiararlo prima di **TipoCoda**)

```
typedef struct Tdato{
   int val;
   //costruttore Default
   Tdato(){  //da implementare }
   //costruttore specifico
   Tdato(int _val){  //da implementare }
   //distruttore
   ~Tdato(){  //da implementare }
   //metodo di stampa
   void print()const{  //da implementare }
} Tdato;
```

Esercizio 1 - Code con array

- Vogliamo scrivere un programma che implementa una **coda** (con un massimo numero di 5 elementi) di tipo **Tdato** usando i vettori

```
CodaPtr C1 = new Coda(5);
```

- Andremo a definire le seguenti **funzioni**:
  - **bool codaIsFull(CodaPtr p)** : controlla se la coda è piena
  - **bool codaIsEmpty(CodaPtr p)** : verifica se la coda è vuota
  - **void put(CodaPtr p, Tdato d)** : controlla se la lista è piena e inserisce x in coda
  - **Tdato get(CodaPtr p)** : estrae l'elemento in testa alla coda
  - **void stampa(CodaPtr p)** : stampa il contenuto della coda
- Creare i file **fifo.h** e **fifo.cpp** che implementano la coda e le funzioni per usarla
- Scrivere un file con il main che prova le funzionalità della coda

Esercizio 1 – esempi

```
Coda c(5);  // stack
c.print();


CodaPtr cp;  // stack
cp = new Coda(5);  // heap
put(cp,d);
d.val=12;
put(cp,d);
//cp->print();
print(cp);
delete cp;


CodaPtr acp[3];  // stack array di 3 puntatori
acp[0] = new Coda(5);   // heap
acp[1] = new Coda(10);  // heap
acp[2] = new Coda(20);  // heap
put(acp[0],d);
//acp[0]->print();
print(acp[0]);

delete acp[0];
delete acp[1];
delete acp[2];
```

Esercizio 1 – main.cpp

```
CodaPtr c1 = new Coda(5);
cout << "La coda e' vuota? " << codaIsEmpty(c1) << endl;
cout << "La coda e' piena? " << codaIsFull(c1) << endl;
cout << "Contenuto della coda:" << endl;
stampa(c1);
cout << "Put 1, ora la coda contiene:" << endl;
put(c1, Tdato(1));
stampa(c1);
cout << "Put 2, 3, 4, 5; ora la coda contiene:" << endl;
put(c1, Tdato(2));
put(c1, Tdato(3));
put(c1, Tdato(4));
put(c1, Tdato(5));
stampa(c1);
cout << "Put 6, ora la coda contiene:" << endl;
put(c1, Tdato(6)); //non viene inserito: coda piena
stampa(c1);
cout << "Get(C1):";
get(c1).stampa();
cout << endl;
stampa(c1);
cout << "Get(C1): “;  get(c1).stampa();  cout << endl;
cout << "Get(C1): ”;  get(c1).stampa();  cout << endl;
```

```
if( !codaIsFull(c1) ) { put(c1, Tdato(2)); }
if( !codaIsFull(c1) ) { put(c1, Tdato(3)); }
if( !codaIsFull(c1) ) { put(c1, Tdato(4)); }
if( !codaIsFull(c1 )) { put(c1, Tdato(5)); }
```

```
if(!codaIsEmpty(c1)){
 cout << "Get(C1): ";
 get(c1).stampa();
 cout << endl;
}
```

Esercizio 1 – main.cpp

```
cout << "Contenuto della coda:" << endl;
stampa(c1);
cout << "Put 6, 7, 8; ora la coda contiene:" << endl;
put(c1,Tdato(6));
put(c1,Tdato(7));
put(c1,Tdato(8));
stampa(c1);
cout << "Svuotiamo la coda:" << endl;
cout << "Get(C1): ";  get(c1).stampa();  cout << endl;
cout << "Get(C1): ";  get(c1).stampa();  cout << endl;
cout << "Get(C1): ";  get(c1).stampa();  cout << endl;
cout << "Get(C1): ";  get(c1).stampa();  cout << endl;
cout << "Get(C1): ";  get(c1).stampa();  cout << endl;
cout << "Contenuto della coda:" << endl;
stampa(c1);
cout << "Get(C1): ”; get(c1).stampa(); cout << endl;
//meglio controllare che la coda non sia vuota prima di estrarre dati
```

```
if( !codaIsFull(c1) ) { put(c1, Tdato(6)); }
if( !codaIsFull(c1) ) { put(c1, Tdato(7)); }
if( !codaIsFull(c1) ) { put(c1, Tdato(8)); }
```

```
if(!codaIsEmpty(c1)){
 cout << "Get(C1): "; get(c1).stampa(); cout << endl;
}
```

Esercizio 1 – fifo.h

```
// Verifica se la coda e' vuota o no
bool codaIsEmpty(CodaPtr p);

// Verifica se la coda e' piena o no
bool codaIsFull(CodaPtr p);

// Inserisce l'elemento d nella coda aumentandone la
   dimensione
void put(CodaPtr p, Tdato d);

// Rimuove un elemento dalla coda, diminuendone la
   dimensione
Tdato get(CodaPtr p);

// Stampa il contenuto della coda
void stampa(CodaPtr p);
```

Esercizio 2 - Code con array

- Utilizzando i files `fifo.h` e `fifo.cpp` definiti, definire un nuovo programma
- Struttura main
  - Definire CodaPtr **w** con dimensione massima 15
  - Ripetere le seguenti operazioni K volte con K dichiarata costante (40)
    - Leggere da tastiera un numero intero
    - Generare un numero intero NI compreso tra 1 e 20
    - Se NI > 10: inserire (PUT) il numero letto da tastiera nella coda
    - Stampare la coda
    - Generare un NUOVO numero intero NI compreso tra 1 e 20
    - Se NI < 10: leggere dalla coda (GET) il valore
    - Stampare la coda

Esercizio 3

- Usare la libreria precedente (`fifo.h` e `fifo.cpp`) in un nuovo programma che:
- Creare 2 code: P e D, entrambe composte da un massimo di 10 elementi
- Genera numeri casuali, fra 1 e 9, e segue le seguenti regole:
  - Se il numero è dispari e < 6, lo mette (se c'è posto) nella coda D
  - Se il numero è pari e < 5 lo mette nella coda P
  - Se il numero è dispari ma > 6, estrae un elemento dalla coda D
  - Se il numero è pari ma > 5, estrae un elemento dalla coda P
- Stampa il contenuto della coda modificata
- Termina dopo aver generato 40 numeri casuali e mostra lo stato finale delle due code

Esercizio 4

- Modificare le funzioni

```
void put(CodaPtr p, Tdato d) { /* codice */ }
Tdato get(CodaPtr p) { /* codice */ }
bool codaIsEmpty(CodaPtr p) { /* codice */ }
bool codaIsFull(CodaPtr p) { /* codice */ }
```

- E trasformarle in metodi della struttura TipoCoda

```
typedef struct TipoCoda {
   [ ... ]

   void put (Tdato d) { /* codice */ }
   Tdato get(){ /* codice */ }
   bool codaIsEmpty()const{ /* codice */ }
   bool codaIsFull()const{ /* codice */ }

   [ ... ]
} TipoCoda;
```

Esercizio 5

- Struttura Tdato: int val;
- Definire nel main un array p di 3 elementi di tipo puntatore a TipoCoda.
  - Ogni elemento dell'array è inizializzato come una coda di dimensione massima 6
- Ripetere per K (10) volte le operazioni
  - Generare un elemento x di tipo Tdato con un valore casuale

    `Tdato dato_casuale()`
  - Selezionare una tra le code dell'array p in modo casuale
  - Se la coda NON è piena
    - Aggiungere x alla coda selezionata
  - Altrimenti (se coda piena)
    - Aggiungere il contenuto di x in file «scarti.txt»

      `void scrivi_file(char nome_file[], Tdato d)`

Esercizio 5

- Se le code non sono tutte vuote

  ```
  bool code_vuote(CodaPtr v[], int dim)
  ```

  - Cercare la coda (indice della coda) che ha più elementi e stampare a video l'indice di tale coda

    ```
    int coda_lunga(CodaPtr v[], int dim)
    ```

  - Cercare la coda (indice della coda) che ha il Tdato da estrarre massimo (Tdato massimo → val massimo) e stampare a video l'indice di tale coda

    ```
    int dato_massimo(CodaPtr v[], int dim)
    ```

- Stampare le code
- De-allocare in modo esplicito le variabili allocate dinamicamente

Esercizio 5

- `Tdato dato_casuale();`
  - Restituisce un dato il cui campo val è inizializzato in modo casuale tra -10 e +100
- `void scrivi_file(char nome_file[], Tdato d)`
  - Input: nome del file su cui stampare e dato da salvare. Aggiunge al contenuto del file indicato il contenuto del campo val di Tdato
  - Un valore per ogni riga
- `bool code_vuote(CodaPtr v[], int dim);`
  - Input: array di puntatori a TipoCoda e sua dimensione effettiva. Restitusice vero se tutte le code dell'array v sono vuote
- `int coda_lunga(CodaPtr v[], int dim);`
  - Input: array di puntatori a TipoCoda e sua dimensione effettiva. Restituisce l'indice della coda dell'array v che ha il dato da estrarre di valore massimo
- `int dato_massimo(CodaPtr v[], int dim);`
  - Input: array di puntatori a TipoCoda e sua dimensione effettiva. Restituisce l'indice della coda dell'array v che ha il dato da estrarre di valore massimo