PILE E CODE

Roberto Nardone, Luigi Romano

ROADMAP

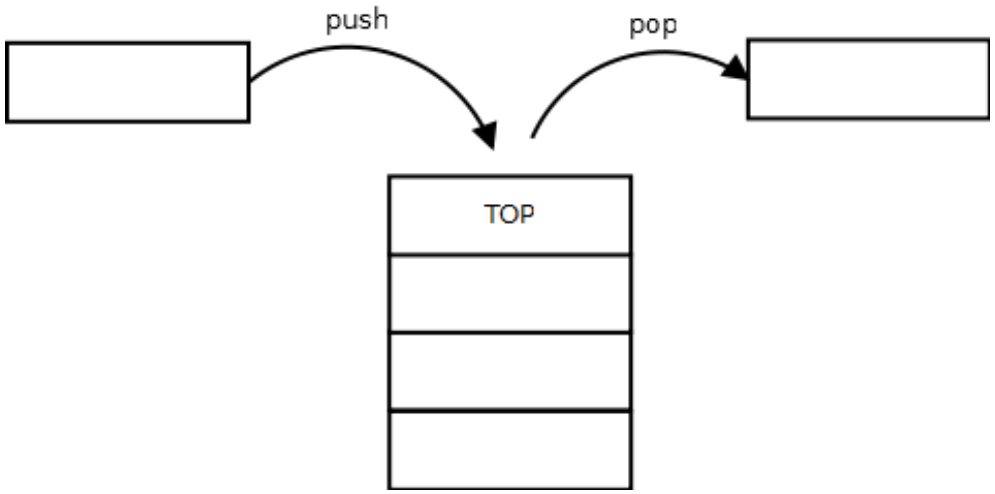
- Struttura dati **LIFO**: Pila
  - Concetti di Base
  - Operazioni sulla Pila
  - Metodo di Implementazione ed Esempio
- Struttura dati **FIFO**: Coda
  - Concetti di Base
  - Operazioni sulla Coda
  - Metodo di Implementazione ed Esempio

Il Concetto di Pila

- Una **pila** (o **stack**) è una lista nella quale si inseriscono e si eliminano elementi solo dalla testa
- L'analogia più comune è quella di una pila di piatti
- Per questo motivo si dice che la pila è gestita con strategia **L**ast **I**n **F**irst **O**ut (**LIFO**)
- Il concetto è che la sequenza di estrazione è esattamente l'inversa di quella di immissione
- Il primo elemento a essere immesso sarà anche l'ultimo a essere estratto

Operazioni sulla Pila

- Le operazioni tipiche di una pila sono:
  - Inserimento di un elemento nella pila (***Push***)
  - Rimozione di un elemento dalla pila (***Pop***)
  - Lettura dell'ultimo elemento inserito senza estrazione (***Top***)
  - Eliminazione dell'elemento in cima alla pila senza leggerlo (***Drop***)



Peculiarità della Pila

- La pila la si può definire con un array. In tal caso la sua dimensione massima è fissa
- Oppure, tramite puntatori e liste dinamiche. In tal caso si usa la memoria dinamica e non vi è alcuna limitazione alla sua dimensione
- Chiaramente può succedere che la pila sia *vuota* o *piena*
- Se il programma tenta di **rimuovere un elemento da una pila vuota** sarà ritornato un errore. Si dice che si verifica il cosidetto ***Stack Underflow***
- Se il programma tenta di **inserire un elemento in una pila piena** sarà ritornato un errore. Si dice che si verifica il cosidetto ***Stack Overflow***

Implementazione della Pila

- Supponiamo di voler definire una pila di interi attraverso un array
- Il metodo migliore di implementazione della Pila è attraverso una classe che conterrà:
  - L'**Array**, membro della classe, rappresentante la pila
  - Una **variabile intera**, membro della classe, rappresentante l'indice alla cima della pila
  - **Costruttore** per inizializzazione dell'indice
  - **Funzioni membro** riguardanti le operazioni sulla pila
- Solitamente si considera il ***fondo*** della pila l'elemento di posizione 0 dell'array
- Dunque alla posizione 1 si inserirà il secondo elemento, e così via

Esempio – Definizione della Classe Pila

- La classe Pila è costituita dunque da:
  - Array *num* avente dimensione massima SIZE
  - Indice alla testa della pila *top*
  - Funzione membro *Push*
  - Funzione membro *Pop*
  - Funzione membro ausiliaria *isEmpty*
  - Funzione membro ausiliaria *isFull*
  - Funzione membro ausiliaria *displayItems*

```
#include<iostream>

#define SIZE 5

using namespace std;

class STACK {
private:
    int num[SIZE];
    int top;
public:
    STACK();    //default constructor
    void push(int);
    int pop();
    bool isEmpty();
    bool isFull();
    void displayItems();
};
```

ESEMPIO – COSTRUTTORE E FUNZIONI AUSILIARIE

- Il costruttore inizializza l'indice alla cima dello stack al valore -1
- La funzione ausiliaria **isEmpty** ritorna 1 (**true**) se la pila è vuota, 0 (**false)** altrimenti
- La funzione ausiliaria **isFull** ritorna 1 (**true**) se l'indice è uguale a SIZE-1 (perchè il conteggio dell'indice parte da zero). Ritorna 0 (**false**) altrimenti
- La funzione ausiliaria **displayItems** stampa a video i valori

```
STACK::STACK() {
    top = -1;
}

bool STACK::isEmpty() {
    if (top == -1)
        return true;
    else
        return false;
}

bool STACK::isFull() {
    if (top == (SIZE - 1))
        return true;
    else
        return false;
}

void STACK::displayItems(){
    int i; //for loop
    cout<<"STACK is: ";
    for(i=(top); i>=0; i--)
        cout<<num[i]<<" ";
    cout<<endl;
}
```

ESEMPIO – FUNZIONI MEMBRO

- Le funzioni membro della classe pila:
  - Push:
    - 1 – Verificare che la pila non è piena
    - 2 – Incrementare di 1 l'indice alla cima della pila
    - 3 – Inserire l'elemento nella posizione dell'indice della pila
  - Pop:
    - 1 – Verificare che la pila non è vuota
    - 2 – Leggere l'elemento alla posizione data dall'indice della pila
    - 3 – Ridurre di 1 l'indice alla cima della pila

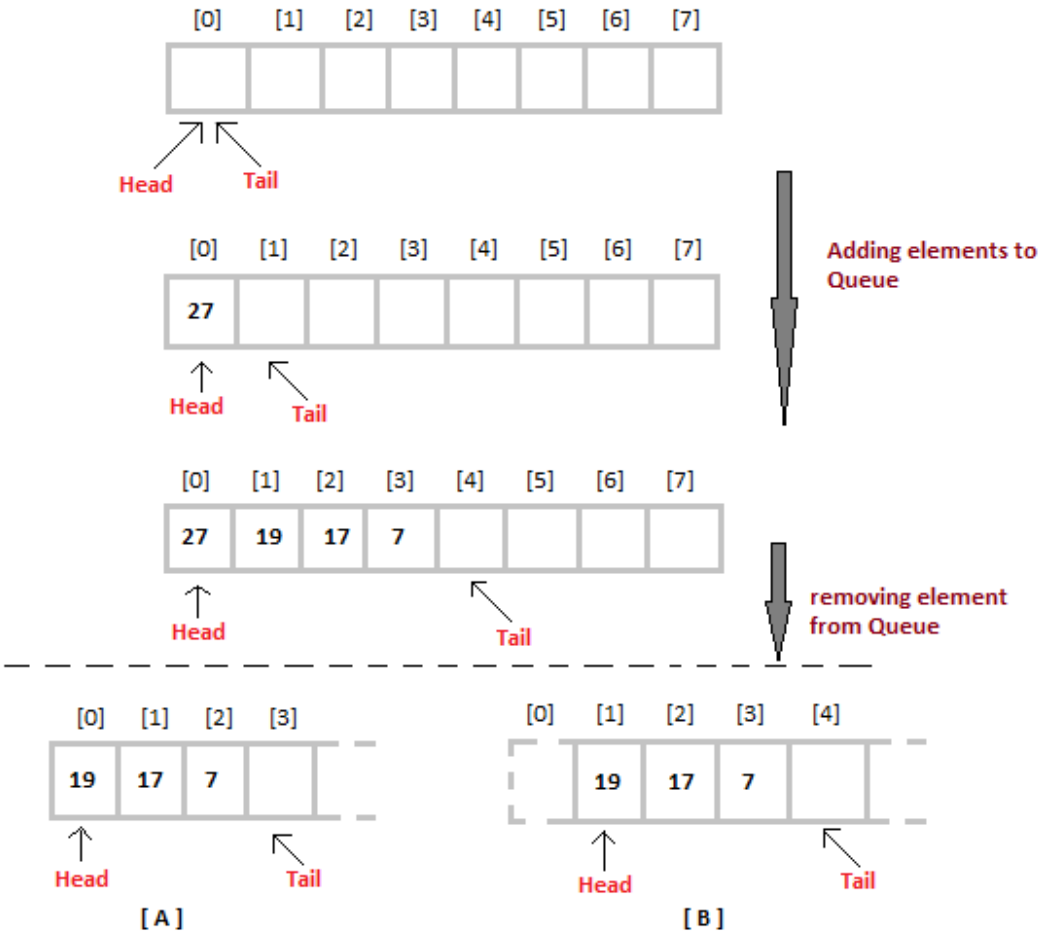
```
void STACK::push(int n) {
    //check stack is full or not
    if (isFull()) {
        cout << "Error: the stack is full" << endl;
        return;
    }
    ++top;
    num[top] = n;
}

int STACK::pop() {
    //to store and print which number
    //is deleted
    int temp;
    //check for empty
    if (isEmpty()) {
        cout << "Error: the stack is empty" << endl;
        return -1;
    }
    temp = num[top];
    --top;
    return temp;

}
```

Il Concetto di Coda

- Una **coda** (o **queue**) è una struttura dati pensata come un vettore che permette di accedere ai dati da ciascuno dei suoi estremi
- L'analogia più comune è quella di una coda di attesa
- Si dice che la coda è gestita con strategia **F**irst **I**n **F**irst **O**ut (**FIFO**)
- Gli elementi si immettono alla fine e si rimuovono dal fronte nello stesso ordine in cui sono stati immessi

Operazioni sulla Coda

- Le operazioni tipiche di una coda sono:
  - Aggiunta di un dato alla fine della coda (*Enqueue*)
  - Eliminazione di un dato dal fronte della coda (*Dequeue*)



Implementazione della Coda attraverso Array – 1/2

- Una possibilità di implementazione della coda è attraverso gli array, come fatto per la pila

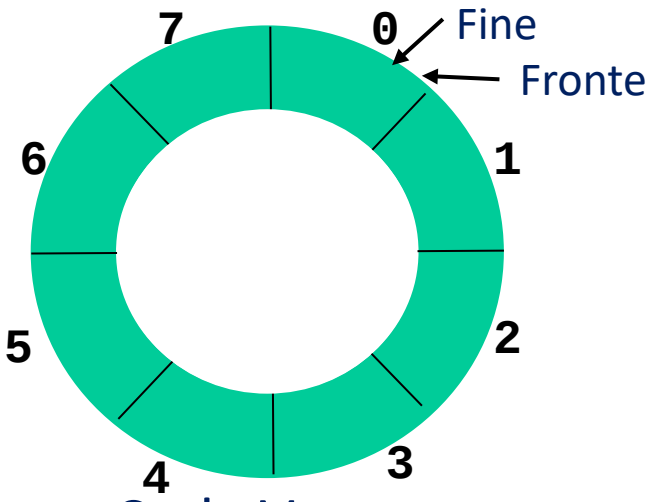
- A differenza della pila, la coda fa uso di due indici per tenere traccia del suo stato:
  - Indice di **fronte** che punta alla testa della coda
  - Indice di **fine** che punta alla prima cella vuota che segue la fine della coda

Implementazione della Coda attraverso Array – 2/2

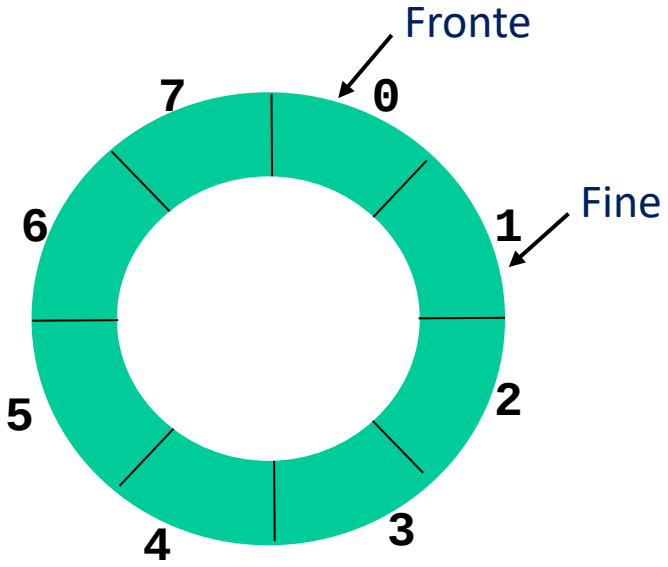
- La coda è considerata vuota quando *fronte* e *fine* hanno lo stesso valore
- Tuttavia, l'uso di un array standard è una soluzione non ottimale da un punto di vista delle performance.
- Che succede se noi inseriamo (*enqueue*) alla posizione 0, e rimuoviamo (*dequeue*) alla posizione *N*?
- Si dovranno far scorrere tutti gli elementi in avanti

Implementazione della Coda attraverso Array Circolare

- La soluzione è dunque quella di fare uso di un array circolare che permette di inserire e rimuovere elementi in un tempo costante
- L'idea dell'array circolare è che, quando si arriva all'ultimo elemento, si ricomincia dal primo.
- Da un punto di vista implementativo, ciò si realizza attraverso l'uso dell'operatore modulo %



Coda Vuota



Coda con un Elemento

Uso dell'Operatore Modulo

- Ricordiamo che l'operatore modulo ritorna il resto della divisione tra i due operandi: 2 % 2 = 0 ; 5 % 2=1
- Nel caso della coda basata su array circolare, l'incremento degli indici di fronte e fine è sempre fatto in modulo pari alla dimensione del vettore
  - Enqueue:
    - Q[fine] = x
      fine= (fine+1) % N
  - Dequeue:
    - x = Q[fronte]
      fronte = (fronte+1) % N

CONDIZIONI DI CODA VUOTA E PIENA

- La condizione di eguaglianza tra indici di coda e testa (Coda==Testa) può indicare sia che la coda è vuota sia che è piena, tuttavia il fatto che la condizione venga a verificarsi a seguito di un inserimento o di un prelievo, può essere sufficiente per disambiguare la condizione
- La coda è piena quando: a seguito di un inserimento in Coda l'indice di coda (Tail) diviene uguale all'indice di testa (Head)
- La coda è vuota quando: a seguito di un prelievo dalla Testa, l'indice di testa diviene uguale all'indice di coda

IMPLEMENTAZIONE DELLA CODA

- Supponiamo di voler definire una Coda di interi attraverso un array
- Il metodo migliore di implementazione della Coda è attraverso una classe che conterrà:
  - L'**Array**, membro della classe, rappresentante la coda
  - Due **variabili intere**, membri della classe, rappresentanti il fronte e la fine coda
  - **Costruttore** per inizializzazione di fronte e fine coda
  - **Funzioni membro** riguardanti le operazioni sulla coda

ESEMPIO – DEFINIZIONE DELLA CLASSE CODA

- La classe Coda è costituita da:
  - Array *A* avente dimensione massima *MAX_SIZE*
  - Indice al testa della coda *T*

  e alla fine della coda *C*
  - Le variabili di stato *piena* e *vuota*
  - Funzione membro *enqueue*
  - Funzione membro *dequeue*
  - Funzione membro *Front*
  - Funzioni membro ausiliarie *isEmpty* e *isFull*
  - Funzione membro ausiliaria *stampa*

```
1  #include <iostream>
2
3  using namespace std;
4
5  const int N = 4;
6  typedef int Elemento;
7
8  class Coda{
9    private:
10     int T,C; //T= Testa C= Coda
11     bool piena, vuota;
12     Elemento c[N];
13   public:
14     Coda();
15     bool isFull();
16     bool isEmpty();
17     Elemento dequeue();
18     void enqueue(Elemento e);
19     void stampa();
20     Elemento front();
21 };
```

ESEMPIO – COSTRUTTORE E FUNZIONI AUSILIARIE

- Il costruttore inizializza *T* e *C* allo stello valore 0; *piena* a *false* e *vuota* a *true*
- Le funzioni ausiliarie isFull e isEmpty restituiscono il valore delle rispettive variabili di stato

```
23  Coda::Coda(){
24      T=C=0;
25      piena = false;
26      vuota = true;
27  }
28  bool Coda::isFull(){
29      return piena;
30  }
31  bool Coda::isEmpty(){
32      return vuota;
33  }
```

ESEMPIO – FUNZIONI MEMBRO

- Le funzioni membro della classe coda:
  - enqueue:
    - 1 – Verificare che la coda non è piena
    - 2 – poiché si sta inserendo un elemento, sicuramente la coda non è vuota
    - 3 – Inserire l'elemento nella posizione dell'indice *C ed* aggiornarlo
    - 4 – se a seguito dell'inserimento C==T allora la coda si riempita
  - dequeue:
    - Se la coda non è vuota, si preleva un elemento dalla posizione T (pertanto la coda sicuramente non è piena essendosi liberata una posizione).
    - Si aggiorna T e se a seguito di ciò diviene pari a C, la coda è vuota

```
34 void Coda::enqueue(Elemento e){
35   if(!piena){
36     vuota = false;
37     c[C] = e;
38     C = (C+1)%N;
39     if(C==T) piena = true;
40   }else {
41     cout << "Coda piena" << endl;
42   }
43 }
```

```
44 Elemento Coda::dequeue(){
45   Elemento e = -1;
46   if(!vuota){
47     piena = false;
48     e = c[T];
49     T = (T+1)%N;
50     if(T==C) vuota = true;
51   }else {
52     cout << "Coda vuota" << endl;
53   }
54   return e;
55 }
```

ESEMPIO – FUNZIONI MEMBRO

- La funzione *front* verifica che la coda non sia vuota e ritorna l'elemento presente nel fronte.

```
57 Elemento Coda::front(){
58     Elemento e = -1;
59     if(!vuota){
60         e = c[T];
61     }
62     return e;
63 }
```

- La funzione *stampa* stampa tutti gli elementi partendo dalla testa e fino a raggiungere la coda

```
64 void Coda::stampa(){
65     if(vuota) cout << "Coda vuota" << endl;
66     else{
67         int i=T;
68         do{
69             cout << c[i] << " ";
70             i=(i+1)%N;
71         }while(i!=C);
72         cout << endl;
73     }
74 }
```

ESERCIZIO

- Implementare sia uno *stack* che una *queue* come mostrato in precedenza e scrivere un main che li usi entrambi