

Realizzazione ADT Coda

Laboratorio di Algoritmi e Strutture Dati

Marco de Gemmis
degemmis@di.uniba.it

**Materiale di base gentilmente concesso
dal dott. Nicola Di Mauro
Ricercatore presso l'Univ. di Bari
Adattamenti di Marco de Gemmis**

ADT Coda

- Gestione dati con criterio FIFO
 - First In First Out
- **E' POSSIBILE AGGIUNGERE ELEMENTI AD UN ESTREMO (“IL FONDO”) E TOGLIERE ELEMENTI DALL'ALTRO ESTREMO (“LA TESTA”)**
- **E' PARTICOLARMENTE ADATTA A RAPPRESENTARE SEQUENZE NELLE QUALI L'ELEMENTO VIENE ELABORATO SECONDO L'ORDINE DI ARRIVO (il primo ad arrivare è quello servito per primo)**

Rappresentazione con vettore circolare: Interfaccia

```
#i f n d e f _ C O D A V T _  
#d e f i n e _ C O D A V T _  
#i n c l u d e < i o s t r e a m >  
#i n c l u d e < c a s s e r t >  
  
t e m p l a t e < c l a s s t i p o e l e m > c l a s s C o d a {  
p u b l i c :  
    C o d a ( i n t );  
    C o d a ();  
    ~ C o d a ();  
    v o i d c r e a C o d a ();  
    b o o l c o d a V u o t a ();  
    v o i d f u o r i C o d a ();  
    t i p o e l e m l e g g i C o d a ();  
    v o i d i n C o d a ( t i p o e l e m );  
p r i v a t e :  
    t i p o e l e m * e l e m e n t i ;  
    i n t t e s t a , l u n g , m a x l u n g ;  
};  
  
// i n t e r f a c c i a
```

Rappresentazione con vettore circolare: implementazione

```
// implementazione

template <class tipoelem> Coda<tipoelem>::Coda(int n){
maxlung = n;
creaCoda();
}

template <class tipoelem> Coda<tipoelem>::Coda(){
maxlung = 100;
creaCoda();
}

template <class tipoelem> Coda<tipoelem>::~~Coda(){delete[] elementi;}

template <class tipoelem> void Coda<tipoelem>::creaCoda(){
elementi = new tipoelem[maxlung];
testa = 0;
Lung = 0; }

template <class tipoelem> bool Coda<tipoelem>::codaVuota(){return (lung == 0);}

template <class tipoelem> tipoelem Coda<tipoelem>::leggiCoda(){
assert (!codaVuota());
return (elementi[testa]);}

template <class tipoelem> void Coda<tipoelem>::fuoriCoda(){
assert(!codaVuota());
testa = (testa + 1) % maxlung;
lung--; }

template <class tipoelem> void Coda<tipoelem>::inCoda(tipoelem a){
assert(lung != maxlung);
elementi[(testa+lung) % maxlung] = a;
lung++;
}
```

Test Coda Interi

```
#include "codavt.h"
#include <iostream>

using namespace std;

int main(){

    Coda<int> C(3);

    C.inCoda(1);
    C.inCoda(2);
    C.inCoda(3);
    // C.inCoda(4); // questo inserimento causa un errore

    cout << " " << C.leggiCoda();
    C.fuoriCoda(); // fuori 1 coda=23
    C.inCoda(4); // coda= 234
    cout << " " << C.leggiCoda();
    C.fuoriCoda(); // fuori 2 coda = 34
    cout << " " << C.leggiCoda();
    C.inCoda(5); // coda = 345
    C.fuoriCoda(); // coda=45
    cout << " " << C.leggiCoda();
    C.fuoriCoda(); // coda =5
    cout << " " << C.leggiCoda();
    C.fuoriCoda(); // coda vuota
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Esercizio svolto#1

- Coda senza duplicati
 - politica “ignora il nuovo elemento”: un elemento non va inserito nella coda se già presente

Svolgimento

```
template <class tipoelem> bool Coda<tipoelem>::contiene(tipoelem e){
// lung è il numero di elementi nella coda
bool trovato = false;
if (!codaVuota())
{
    int i=0;
    int indice;
    while (!trovato && i<lung)
    {
        indice = (testa + i) % maxlung;
        if (elementi[indice] == e)
            trovato=true;
        i++;
    }
}
return trovato;
}
```

```
template <class tipoelem> void Coda<tipoelem>::inCodaNDup(tipoelem a){
assert(lung != maxlung);
if (!contiene(a))
{
    elementi[(testa+lung) % maxlung] = a;
    lung++;
}
}
```

Stampa

```
template <class tipoelem> void Coda<tipoelem>::stampaCoda(){  
// lung è il numero di elementi nella coda  
int indice;  
for (int i=0; i<lung;i++)  
{  
    indice = (testa + i) % maxlung;  
    cout << elementi[indice] << " ";  
}  
}
```


Test

```
#include "codavt.h"
#include <iostream>

using namespace std;

int main(){

    Coda<char> C(6);

    C.inCodaNDup('X');
    C.inCodaNDup('Y');
    C.inCodaNDup('Z');
    C.inCodaNDup('A');
    C.inCodaNDup('B');
    C.stampaCoda();
    cout << "\ninserisco duplicati\n";
    C.inCodaNDup('X');
    C.inCodaNDup('Y');
    C.inCodaNDup('Y');
    C.inCodaNDup('X');
    C.inCodaNDup('A');
    C.fuoriCoda();
    C.inCodaNDup('D');
    C.fuoriCoda();
    C.inCodaNDup('D');
    cout << "\n... e ristampo coda\n";
    C.stampaCoda();
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Esercizio svolto#2

Simulare una situazione in cui si assegnano in modo casuale $N=10$ utenti in attesa di servizio, identificati da un numero intero, a una di $M=3$ possibili code.

Dopo che $N/2$ utenti sono stati accodati, ogni volta che un nuovo utente è accodato, si sceglie una coda a caso e l'utente in testa alla coda è servito.

Si stampi:

- il contenuto delle code dopo la sistemazione dei primi $N/2$ utenti;
- l'utente accodato, quello servito e lo stato delle code a partire da quando gli utenti sono serviti fino ad esaurimento delle code

Svolgimento

```
#include "codavt.h"
#include <iostream>

using namespace std;

int main(){

static const int M = 3;
static const int N = 10;
// semplificazione: N pari!
Coda<int> code[M];
// assegnazione N/2 utenti
int middle=N/2;
for (int i=1; i<=middle; i++)
{
    int in = rand() % M;
    code[in].inCoda(i);
    cout << "inserito utente: " << i << " in coda: " << in << endl;
}
cout << "SITUAZIONE CODE DOPO INSERIMENTI" << endl;
for (int k=0; k<M; k++)
{
    code[k].stampaCoda();
    cout << endl;
}
```

Svolgimento

```
for (int i=middle+1; i<=N; i++)
{
    // sistemiamo gli altri utenti e nel contempo serviamo i primi N/2 accodati
    int in = rand() % M;
    cout << "Accodato Utente: " << i << " in coda: " << in << endl;
    code[in].inCoda(i);
    int out = rand() % M;

    while (code[out].codaVuota())
        out= rand() % M;
    cout << "Servito Utente: " << code[out].leggiCoda() << endl;
    code[out].fuoriCoda();
    for (int k=0; k<M; k++)
    {
        code[k].stampaCoda();
        cout << endl;
    }
}
```

Svolgimento

```
// ora serviamo gli ultimi fino ad esaurimento
cout << "ORA SERVIAMO AD ESAURIMENTO" << endl;
bool continua = false;
for (int i=0; i<M; i++)
    continua = (continua || !code[i].codaVuota());
while (continua)
{
    int out= rand() % M; // cerca una coda non vuota;
    while (code[out].codaVuota())
        out= rand() % M;
    cout << "Servito Utente: " << code[out].leggiCoda() << endl;
    code[out].fuoriCoda();
    for (int k=0; k<M; k++)
    {
        code[k].stampaCoda();
        cout << endl;
    }
    continua = false;
    for (int i=0; i<M; i++)
        continua = (continua || !code[i].codaVuota());
}
system("PAUSE");
return EXIT_SUCCESS;
}
```