Fondamenti di Informatica C

Franco Zambonelli

LE CODE

Coda (First-in-first-out)

La coda è una lista in cui:
- l'inserimento avviene ad una estremità (coda),
- mentre cancellazione e prelievo di dati avvengono all'altra estremità (testa).

Gli elementi vengono prelevati nello stesso ordine in cui sono stati inseriti.

Operazioni base su coda

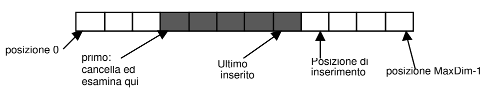
descrizione	*intestazione delle funzioni in C*
crea una coda vuota	`void CodaCrea(Coda *C);`
inserisce un nuovo atomo in fondo	`int CodaIn(Coda *C, Atomo A);`
cancella l'atomo in testa	`int CodaOut(Coda *C, Atomo *A);`
vero se la coda è vuota	`int CodaVuota(Coda *C);`

Implementazione di coda con array

conviene memorizzare gli elementi in posizioni consecutive, ma non necessariamente a partire dalla prima
- quando la coda è stata appena creata, il primo inserimento avviene nella prima posizione
- gli inserimenti consecutivi di seguito, in posizioni verso destra
- le cancellazioni avverranno a partire dalla posizione 1 e via via opereranno su una posizione che si sposta verso destra

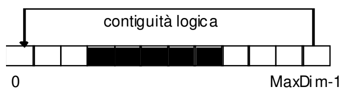
S crea un *pacchetto* di dati validi che si sposta verso destra.

Bisogna tenere traccia di due posizioni:
- primo elemento da estrarre
- del punto di inserimento



Gestione circolare di array

Quando la coda raggiunge l'ultima posizione dell'array, è necessario gestire l'array come se si trattasse di una struttura circolare:
- la posizione successiva a `MaxDim-1` é `0`.



Si modifica il meccanismo di incremento di indice:
- `PosizSucc = Posiz % MaxDim`,
- genera la successione `0, 1, 2, ..., MaxDim-1, 0, 1, 2, ....`

Inoltre:
- è necessario sapere quando la coda è piena o vuota.
- Se *posizione del primo* = *posizione di inserimento* la coda può essere sia piena che vuota
- Serve informazione supplementare *vera* quando la coda è vuota

```
/*  CodeArr.h:  coda  implementata  con  array
circolare  */
/*  Versione con accesso distruttivo
                           */

#define CodaStatoPiena 2
#define CodaStatoVuota 1
#define CodaOK 0

typedef int Posiz;

typedef struct {
       int Vuoto; /*  vero  quando  coda  vuota
*/
       Posiz   Primo,   /*   Posizione   primo
inserito */
               Inser;     /*   Posizione   del
prossimo da inserire */
       Atomo   Dati[MaxDim];
} Coda;

extern void CodaCrea(Coda *C);
extern int CodaVuota(Coda *C);
extern Posiz Primo(Coda *C);
extern int CodaIn(Coda *C, Atomo A);
extern int CodaOut(Coda *C, Atomo *A);
extern char *CodaErrore ();
extern int CodaStato;/* variabile di stato */
```



```
/* CodaArr.c   */
#include "Infobase.h"
#include "CodaArr.h"

void CodaCrea(Coda *C){
    C->Vuoto=1;
    C->Primo=0;
    C->Inser=0;
}                          /* end CodaCrea */
int CodaIn(Coda *C, Atomo A){
  if ( (C->Inser==C->Primo) && (!(C->Vuoto)))
     CodaStato = CodaStatoPiena;
  else {
   CodaStato = CodaOK;
   C->Dati[C->Inser] = A;
   C->Inser = C->Inser%MaxDim;
   C->Vuoto=0;
  }
  return CodaStato;
} /* end CodaIn */
```



```
int CodaOut(Coda *C, Atomo *A){
  if (C->Vuoto)
     CodaStato = CodaStatoVuota;
  else {
   CodaStato = CodaOK;
   *A=C->Dati[C->Primo];
   C->Primo = C->Primo%MaxDim;
   if (C->Primo==C->Inser)
     C->Vuoto=1;
  }
  return CodaStato;
} /* end CodaOut */
int CodaVuota(Coda *C){ /* *C per economia */
  return C->Vuoto;
}    /* end CodaVuota */
char *CodaErrore (){
  switch(CodaStato){
   case  CodaStatoVuota :
     return "Errore: Coda vuota";
   break;
   case  CodaStatoPiena :
     return "Errore: Coda piena";
  }
  return "Variabile di stato errata";
}  /* end CodaErrore */
```

Complessità computazionale delle operazioni di coda su array:

- costante (no cicli o ricorsioni!)

Si giustifica, vedendo la coda come un tipo particolare lista in cui inserimenti e le cancellazioni avvengono sempre a una estremità

Implementazione di coda con puntatori

Struttura degli elementi è identica a quella della lista.

Sono inoltre necessari un puntatore al primo elemento e uno alla posizione di inserimento.

NON C'E' LIMITE AL NUMERO DI ELEMENTI!!!

```
/* CodaPun.h */
#define CodaStatoVuota 1
#define CodaOK 0

typedef struct  TCella{
 struct TCella  *Prox;
 Atomo Dato;
}Cella;

typedef  Cella *Posiz;

typedef struct{
 Posiz Primo,Ultimo;
} Coda;

extern void CodaCrea(Coda *C);
extern int CodaVuota(Coda *C);
extern int CodaIn(Coda *C, Atomo A);
extern int CodaOut(Coda *C, Atomo *A);
extern char *CodaErrore ();
extern int CodaStato;
```

```
/* CodaPun.c */
#include "Infobase.h"
#include "CodaPun.h"
#include <stdlib.h>

int CodaStato=0;

/* CodaCrea */
void CodaCrea(Coda *C){
 C->Primo=NULL;
 C->Ultimo=NULL;
 CodaStato = CodaOK;
} /* end CodaCrea */

/* CodaIn */
int CodaIn(Coda *C, Atomo A){
 Posiz Temp;
 CodaStato = CodaOK;
 Temp=malloc(sizeof(Cella));
 Temp->Dato = A;
 Temp->Prox = NULL;
 if (C->Ultimo==NULL)    C->Primo = Temp;
 else
   C->Ultimo->Prox = Temp;
 C->Ultimo = Temp;
 return CodaStato;
} /* end CodaIn */
```



```
/* CodaOut */
int CodaOut(Coda *C, Atomo *A){
 Posiz Temp;
 if (C->Primo==NULL)
    CodaStato = CodaStatoVuota;
 else {
   Temp=C->Primo;
   if (C->Primo==C->Ultimo)
    C->Ultimo=NULL;
   C->Primo = C->Primo->Prox;
   *A=Temp->Dato;
   free(Temp);
   CodaStato = CodaOK;
 }
 return CodaStato;
} /* end CodaOut */

/* CodaVuota */
int CodaVuota(Coda *C){ /* *C per economia */
 return (C->Primo==NULL);
} /* end CodaVuota */
```

Complessitá computazionale: ancora costante!

Utilizzo della struttura dati Coda

Il seguente programma mostra un utilizzo della struttura coda, effettuando inserimenti ed estrazioni a comando.

Per il dato elementare, a scopo dimostrativo si utilizza un semplice intero (si può fare riferimento alla libreria InfoBase)

L'algoritmo del programma di prova è il seguente:

```
- inizializza la coda
- ripeti
  - prendi un comando da input
  - se il comando è
    - inserimento
      - ripeti
        - prendi un intero da input
        - inseriscilo
        - in caso di insuccesso: messaggio di
errore
        finché l'intero è diverso da zero
    - estrazione
      - estrai e mostra
      - in caso di insuccesso: messaggio di
errore
  finché comando è diverso da fine
```



```
#include "Infobase.h"
#include "CodaArr.h" /* Sostituibile con
CodaPun.h */
#define Inserimento 'I'
#define Estrazione 'E'
#define Fine 'F'

int main(){
 Coda C;
 Atomo A;
 char Comando[80]; /* per contenere una
intera
                   linea di Comando */

 CodaCrea(&C);
```

```
  do{
   printf("Comando? ");
   gets(Comando); /* carica la linea di
comando,
              considerando solo il primo
carattere */
   if (Comando[0]>='a') Comando[0]-=32;
                        /* converte in
maiuscolo */
   switch (Comando[0]){
     case Inserimento :
  printf("inserimento\n");
            do{
              Acquisisci(&A);
              if(!AtomoNullo(A))
                if(CodaIn(&C,A)!=CodaOK)
                  printf("coda piena\n");
            } while(!AtomoNullo(A));
            break;
     case Estrazione : if
(CodaOut(&C,&A)!=CodaOK)
              printf("coda vuota\n");
            else Visualizza(A);
            break;
     case Fine:   break;
     default : printf("Comando errato\n");
   }
  } while (Comando[0]!=Fine);
   printf("Fine                    lavoro\n");
   return                                 0;
}
```