UNIVERSITÀ
degli STUDI
di CATANIA

Fondamenti di Informatica

Corso di Laurea in Ingegneria Elettronica

Corso di Laurea in Ingegneria Informatica

Struttura Dati Coda: Teoria

La presente dispensa è stata prodotta dal Prof.S.Cavalieri, ed è ufficialmente adottata dai corsi di Fondamenti di Informatica per Ingegneria Informatica/Elettronica

Indice

1. La Struttura Dati Astratta (ADT) Coda

La coda o queue è una struttura dati astratta (ADT= Abstract Data Type) composta da un insieme di elementi omogenei tra loro, ossia appartenenti allo stesso tipo. La caratteristica principale di tale insieme è che ogni elemento occupa una determinata posizione nell'insieme e che l'inserimento di un nuovo elemento e l'estrazione di un elemento esistente sono consentite solo in specifiche posizioni; in particolare, l'inserimento è consentito solo ad una determinata posizione della coda, detta **rear**, e l'estrazione è consentita solo per un'altra particolare posizione, detta **front**. Questa gestione è anche chiamata FIFO (First In First Out), in quanto il primo elemento inserito è anche il primo ad essere estratto.

Sia data una coda vuota, ossia priva di elementi. Si supponga di inserire un elemento denominato a_1; esso rappresenterà contemporaneamente l'elemento di posizione **rear** e **front** della coda, visto che è l'unico elemento della coda. Adesso si supponga di inserirne un altro, a_2; esso rappresenterà il nuovo elemento di posizione **rear** della coda, mentre il **front** sarà sempre relativo all'elemento a_1. La coda attualmente è così costituita:

$$a_1, a_2$$

Adesso si supponga di inserirne un altro, a_3. Esso verrà a rappresentare il nuovo elemento di posizione **rear** della coda, mentre il **front** sarà sempre rappresentato da a_1. La coda sarà così composta:

$$a_1, a_2, a_3$$

Supponendo, in generale, che la coda sia composta dagli elementi a_1, a_2, a_3, …., a_n, l'inserimento di un elemento x produce la nuova coda a_1, a_2, a_3, …., a_n, x. L'elemento di posizione **rear** diviene x. L'estrazione di un elemento della coda si riferisce sempre all'elemento di posizione **front**, ossia a_1; dopo l'operazione di estrazione la coda diviene a_2, a_3, …., a_n, x. Non è possibile estrarre alcun altro elemento della coda eccetto l'elemento di posizione **front**.

2.Operazioni Primitive sull'ADT Coda

La struttura dati Coda è caratterizzata da alcune operazioni fondamentali, che nel seguito verrano illustrate. Sia:

1. q una generica coda
2. x un generico elemento dello stesso tipo degli elementi presenti nella Coda q.

Nel seguito il tipo degli elementi contenuti in una coda verrà chiamato **tipobaseQueue.**

Le seguenti operazioni primitive vengono definite per un ADT Coda:

Struttura Dati Coda

- ***MakeNullQueue(q)***. La funzione restituisce una Coda inizializzata, ossia una coda vuota pronta ad accogliere nuovi elementi.
- ***EmptyQueue(q).*** La funzione restituisce il valore booleano vero se la Coda è vuota.
- ***FullQueue(q)***. La funzione restituisce il valore booleano vero se la Coda è piena, ossia se non è possibile inserire altri elementi.
- ***EnQueue(q,x)***. Questa funzione inserisce l'elemento x nella posizione rappresentata da **rear**. Se q è la Coda a_1, a_2,…, a_n, e se il rear è rappresentato dalla posizione dell'elemento a_n, l'inserimento dell'elemento x avviene in posizione successiva a tale elemento, e la funzione ***EnQueue(q,x)*** restituisce la nuova Coda a_1, a_2,…, a_n, x.
- ***DeQueue(q)***. Questa funzione elimina l'elemento della coda in posizione **front**. Se q è la Coda a_1, a_2,…, a_n e il **front** è rappresentato dalla posizione dell'elemento a_1, la chiamata alla funzione ***DeQueue(q)*** restituisce la nuova coda a_2,a_3,…, a_n e il nuovo front è rappresentata dalla posizione dell'elemento a_2.
- ***Front(q)***. La funzione restituisce il primo elemento della coda (l'elemento viene solamente letto, non viene estratto).

3.Rappresentazione della Struttura Dati Coda

Per la Coda verranno considerate due rappresentazioni: quella sequenziale con vettore e quella collegata con puntatori.

3.1.Rappresentazione Sequenziale

La rappresentazione sequenziale di un ADT Coda con vettore può essere realizzata come mostrato dalla seguente Figura 1, dove viene supposto che la coda contenga elementi di tipo intero e sia composta dagli elementi: 23, 35, 52, 87.

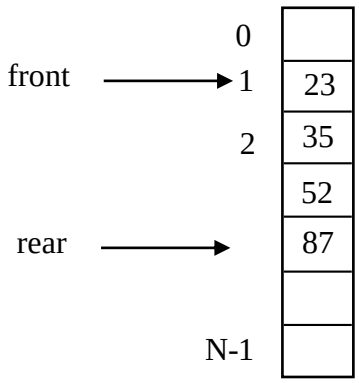

Figura 1 - Rappresentazione sequenziale tipo coda

Come si vede, oltre ad utilizzare un vettore di elementi di tipo tipobaseQueue, vengono utilizzate due variabili, **front** e **rear**, che indicano rispettivamente la posizione del primo elemento della coda e dell'ultimo elemento della coda.

Inizialmente, quando la coda è vuota si assume che **front=rear=-1**.

Quando viene inserito il primo elemento, **rear** viene posto a 0 (ossia viene incrementato) e l'elemento viene inserito nel vettore nella componente di indice **rear** (ossia 0). Anche **front** viene assegnato a 0, in quanto è necessario rappresentare la presenza dell'unico elemento in coda. Dunque dopo l'inserimento del primo elemento in coda, **front** e **rear** saranno uguali e pari a 0, che rappresenta l'indice del vettore in cui è memorizzato l'unico elemento della coda. In questa implementazione infatti si suppone gli elementi della coda vengono inseriti nel vettore a partire dall'indice 0.

Quando un nuovo elemento deve essere inserito successivamente, **rear** viene incrementato di 1 e l'elemento viene inserito nel vettore nella casella di indice fornito da **rear**. Si noti che l'incremento dell'indice **rear** è circolare, come è mostrato nella seguente Figura 2:

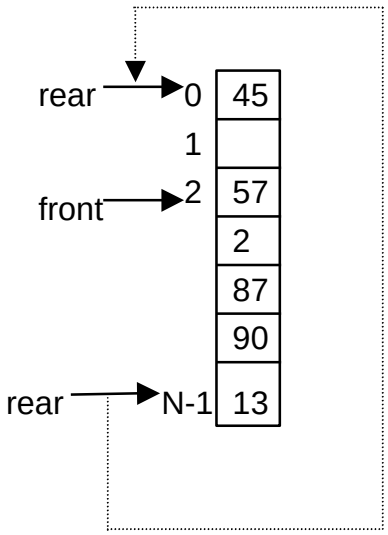

Figura 2 – Incremento circolare dell'indice rear

Nella Figura 2 è chiaro che, prima dell'inserimento, **rear** corrisponde all'ultimo elemento del vettore. Il nuovo elemento può essere inserito nella prima posizione del vettore, ottenuta incrementando **rear** nella seguente maniera:

rear=(rear+1) % N, dove N è la dimensione del vettore

L'inserimento di un elemento è possibile solo se la coda non è piena. La condizione di coda piena è rappresentata in Figura 3:

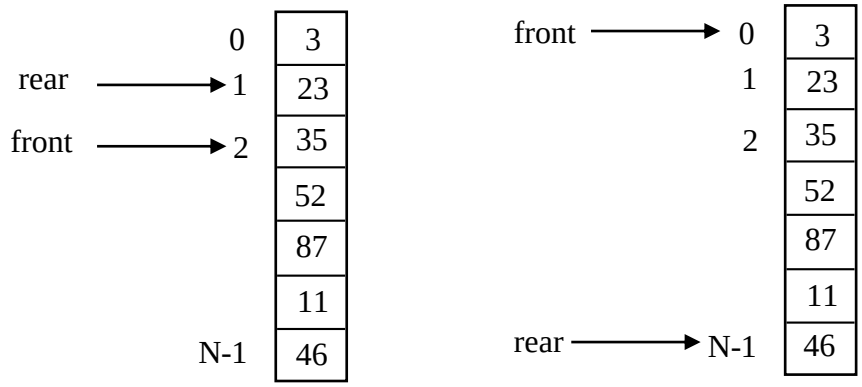

Figura 3 – Condizioni di Coda Piena

In questi due esempi mostrati dalla Figura 3, è chiaro che il valore di **rear** non può essere più incrementato per inserire un nuovo elemento. La condizione di coda piena è dunque:

front = = (rear+1)%N

Per quanto riguarda la cancellazione dell'elemento di posizione **front**, essa viene semplicemente realizzata incrementando la variabile **front**. Dopo aver effettuato la cancellazione, è ovviamente necessario verificare che la coda non diventi vuota, perché in tal caso bisogna aggiornare la variabile **rear**. Se la coda diviene vuota, sia **front** che **rear** divengono pari a -1. La condizione di coda vuota viene controllata prima della cancellazione dell'elemento **front**. Se **front** e **rear** sono coincidenti, significa che la coda possiede un solo elemento, cioè quello che deve essere eliminato. Ciò significa che dopo l'inserimento la coda diverrà vuota.

L’incremento della variabile **front** deve avvenire in modo circolare, analogamente a quanto visto per la variabile **rear**. Ad esempio si consideri la seguente cancellazione dell’elemento 13 mostrato dalla Figura 4.

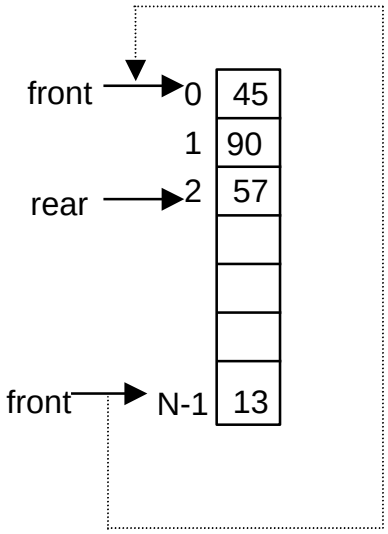

Figura 4 – Incremento circolare della variabile front

Analogamente a quanto visto per la variabile **rear**, l'incremento della variabile **front** deve avvenire nella seguente maniera:

front=(front+1) % N, dove N è la dimensione del vettore

Nel seguito verrà mostrata la rappresentazione sequenziale in C di un ADT Coda mediante il tipo di dato vettore. Nella rappresentazione in C verrà utilizzato un tipo di nome **tipobaseQueue** per rappresentare il generico tipo di dato di ogni singolo elemento contenuto nella coda. La definizione del tipo tipobaseQueue avverrà in C tramite lo strumento typedef. Ad esempio se il tipo di dato di ogni singolo elemento fosse float, l'implementazione del tipo tipobaseQueue risulterebbe:

```
typedef float tipobaseQueue;
```

Visto che molte funzioni primitive sulla Coda restituiscono un valore booleano, converrà definire il tipo **boolean**, definito come alias del tipo di dato short, ossia:

```
typedef short boolean;
```

La coda verrà rappresentata in C da una struttura (struct), di nome ***queue***, composta da tre campi: il vettore info che contiene gli elementi della coda, gli interi **front** e **rear** che individuano la posizione del primo e dell'ultimo elemento della coda. La sua implementazione è:

```
#define N 10

typedef struct {
  tipobaseQueue info[N];
  int front,rear;
} queue;
```

Struttura Dati Coda

Nell'implementazione in C della struttura coda, sarà comodo (e non necessario) l'utilizzo di una costante per rappresentare la condizione di coda vuota. Tale condizione si verifica quando **front** e **rear** assumono il valore -1. La condizione di coda vuota può essere, dunque, rappresentata da una costante, denominata **CODAVUOTA**, pari al valore –1. La costante CODAVUOTA può essere rappresentata dalla seguente definizione:

```
#define CODAVUOTA –1
```

Rappresentazione delle Operazioni Primitive

Le seguenti procedure e funzioni realizzano le operazioni primitive della coda:

```
void MakeNullQueue(queue *q) {
  q->front=q->rear=CODAVUOTA;
}

boolean EmptyQueue(queue q) {
  return(q.front==CODAVUOTA);
}

boolean FullQueue(queue q) {
  return(q.front==((q.rear+1)%N));
}

void EnQueue(queue *q, tipobaseQueue x) {
  if (!FullQueue(*q)) {
    q->rear=(q->rear+1)%N;
    q->info[q->rear]=x;
    if (q->front==CODAVUOTA) q->front=q->rear;
  }
}

void DeQueue(queue *q) {
  if (!EmptyQueue(*q)) {
    if (q->front==q->rear) MakeNullQueue(q);
    else q->front=(q->front+1)%N;
  }
}

tipobaseQueue Front(queue q) {
  if (!EmptyQueue(q))
      return(q.info[q.front]);
}
```

3.2.Rappresentazione Collegata con Puntatori

La rappresentazione collegata con puntatori si basa sull'idea di considerare elementi allocati dinamicamente in memoria RAM (ossia nella memoria Heap), ciascuno dei quali è rappresentato da due campi. Un campo memorizza il generico elemento della coda di tipobaseQueue, mentre l'altro campo memorizza la posizione del successivo elemento della coda. Si noti che gli elementi della coda, vengono allocati nell'Heap e dunque potranno essere sparsi in memoria e non per forza

occupare posizioni sequenziali (a differenza dell'implementazione sequenziale con vettore). Per tale motivo è necessario per ogni elemento, memorizzare l'indirizzo del successivo.

Ad esempio, sia data la struttura dati coda composta dagli elementi: a_1, a_2, a_3, a_4, ……, a_n . Essa può essere rappresentata dalla sequenza di elementi allocati in memoria RAM, mostrata in Figura 5.

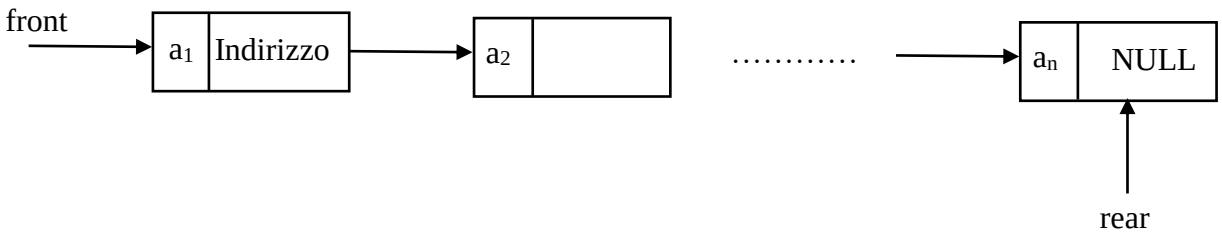

Figura 5 – Rappresentazione collegata di una Coda

Come si vede ogni elemento della coda ha due campi: l'elemento della coda e l'indirizzo (puntatore) all'elemento successivo della coda. L'ultimo elemento conterrà il valore NULL come indirizzo del successivo; tale valore permette di far comprendere che l'elemento corrispondente è l'ultimo della coda ossia non ha un successore. In questa rappresentazione collegata, vengono sempre utilizzate due variabili **front** e **rear** che però rappresentano i puntatori al primo e all'ultimo elemento della coda, rispettivamente.

L'inserimento di un nuovo elemento viene realizzato attraverso il puntatore **rear**, come mostrato dalla Figura 6. Come visibile, **rear** assume il valore dell'indirizzo del nuovo elemento **x**, e il campo puntatore dell'elemento che prima dell'inserimento era l'ultimo (ossia $\mathbf{a_n}$), assume anch'esso il valore dell'indirizzo del nuovo elemento. Ovviamente il campo puntatore del nuovo elemento (ossia il campo che contiene l'indirizzo del successivo elemento) inserito viene posto a NULL.

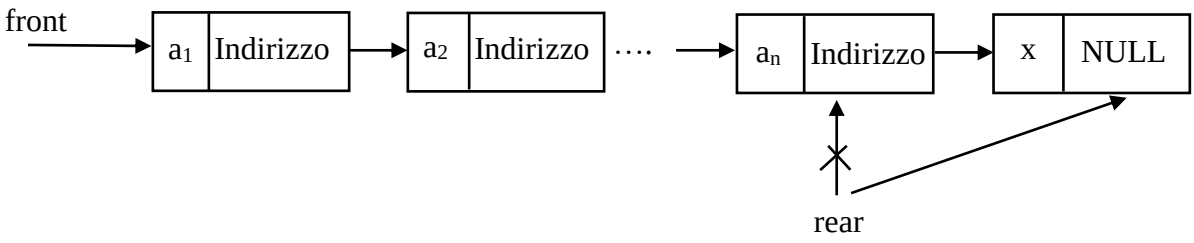

Figura 6 – Inserimento in Coda

Nel caso la coda fosse vuota, tale condizione viene espressa dal valore NULL per entrambe le variabili **front** e **rear**. Nel caso di inserimento in una coda vuota, entrambe le variabili **front** e **rear** assumono l'indirizzo dell'unico elemento inserito, come mostrato dalla Figura 7.

Struttura Dati Coda

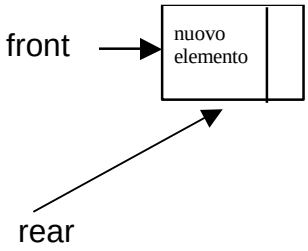

Figura 7 – Inserimento in Coda vuota

L'estrazione di un elemento dalla coda viene realizzato tramite il puntatore front, come visibile dalla Figura 8.

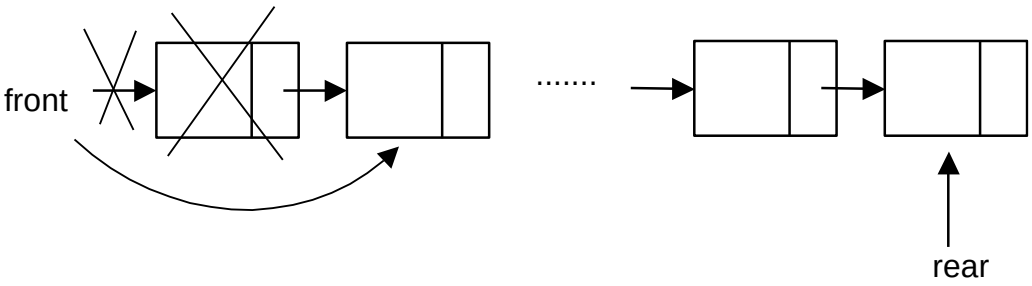

Figura 8 – Estrazione dalla coda

Come si vede, il primo elemento viene eliminato e il valore di front assume l'indirizzo del secondo elemento. Ovviamente nel caso in cui la coda ha un solo elemento, l'estrazione del primo elemento comporterà che sia **front** sia **rear** assumino il valore di NULL, al fine di indicare nuovamente la condizione di coda vuota.

La precedente rappresentazione può essere definita in C nella seguente maniera.

Innanzitutto, come già detto in precedenza, si farà uso del tipo **tipobaseQueue**, che rappresenta il tipo di dato di ogni singolo elemento contenuto nella coda. Il tipo tipobaseQueue verrà sempre definito in C tramite il comando typedef. Ad esempio se il tipo di dato di ogni singolo elemento fosse uno struct contenente i campi: cognome, nome, età, l'implementazione del tipo tipobaseQueue risulterebbe:

```
#define S 15

typedef struct {
  char cognome[S], nome[S];
  unsigned short eta;
}tipobaseQueue;
```

La struttura dati Coda è rappresentata dal tipo ***queue*** (mostrato nel seguito) che è uno struct che contiene i puntatori al primo e all'ultimo elemento della coda stessa. Tali puntatori sono **front** e **rear**, come detto. Essi puntano ad elementi di tipo ***struct nodoQueue***, composti da due campi: info e next.

Il campo info contiene un elemento di tipo tipobaseQueue, mentre il campo next contiene il puntatore all'elemento successivo. L'implementazione della struttura dati Coda è composta, dunque, dalle seguenti due strutture:

```
typedef struct nodo{
 tipobaseQueue info;
 struct nodo *next;
} nodoQueue;

typedef struct {
 nodoQueue * front, * rear;
} queue;
```

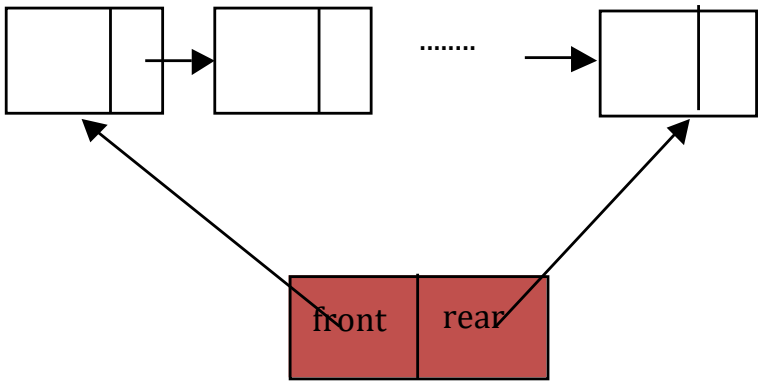

Figura 9 – Implementazione della coda con puntatori

Nell'implementazione in C della Coda con puntatori, si farà uso di una costante per rappresentare la condizione di coda vuota. Tale condizione si verifica quando il puntatore al primo elemento della coda è NULL. Verrà, dunque, definita una costante, di nome CODAVUOTA, che può essere rappresentata da:

```
#define CODAVUOTA NULL
```

La condizione di coda vuota verrà testata dalla funzione booleana EmptyQueue, che ritornerà il valore vero se il puntatore al primo elemento della coda è NULL, ossia CODAVUOTA. La condizione di coda piena viene testata dalla funzione booleana FullQueue; in base a come è stata fatta l'implementazione della coda, tale condizione si realizza solo quando lo spazio di allocazione nell'Heap è insufficiente o per qualunque motivo non sia più possibile allocare memoria nell'Heap. Nell'implementazione presentata in questa dispensa, ciò verrà verificato direttamente dalla funzione EnQueue in base al risultato della funzione malloc(), che la funzione EnQueue deve necessariamente

invocare per inserire un nuovo elemento in coda. Se la funzione malloc() restituisce il valore NULL, allora significa che non è possibile più allocare elementi in Heap e dunque l'operazione di inserimento in coda non verrà effettuata. Per tale motivo, nel seguito la funzione FullQueue non verrà utilizzata per testare se l'Heap è pieno o in genere non è possibile più allocare memoria nell'Heap e verrà implementata semplicemente come una funzione che torna sempre falso (0).

Rappresentazione delle Operazioni Primitive

Le seguenti procedure e funzioni realizzano le operazioni di base della Coda.

```
void MakeNullQueue(queue *q){
  q->front=q->rear=CODAVUOTA;
}

boolean EmptyQueue(queue q){
 return(q.front==CODAVUOTA);
}

boolean FullQueue(queue q){
 return(0);
}

void EnQueue(queue *q, tipobaseQueue x){
  nodoQueue * temp;

  temp=(nodoQueue *)malloc(sizeof(nodoQueue));
  if (temp != NULL) {
      temp->info=x;
      temp->next=CODAVUOTA;
      if (EmptyQueue(*q)) q->front=q->rear=temp;
      else {
           q->rear->next=temp;
           q->rear=temp;
      }
  }
}


void DeQueue(queue *q){
  nodoQueue * temp;
  if (!EmptyQueue(*q)) {
       temp=q->front->next;
       free (q->front);
       q->front=temp;
       if (q->front==CODAVUOTA) q->rear=CODAVUOTA;
  }
}


tipobaseQueue Front(queue q){
  if (!EmptyQueue(q))
             return(q.front->info);
}
```

4. Esempio di Programma di Gestione di una Struttura Dati Coda.

Il seguente esempio di programma si riferisce alla gestione delle funzioni principali di una struttura dati Coda tramite un menu di comandi.

Nel programma vengono utilizzate due funzioni dipendenti dall'implementazione del tipobaseQueue. Le funzioni sono:

- LeggiElementoQueue, che legge da tastiera il valore che deve assumere un nuovo elemento da inserire nella struttura dati Coda;
- VisualizzaElementoQueue, che visualizza sul video un elemento della struttura dati Coda.

Queste funzioni sono strettamente legate alla definizione del *tipobaseQueue*. Ad esempio supponendo di implementare il tipobaseQueue tramite un float, si dovrebbero aggiungere al programma le seguenti funzioni:

```
typedef float tipobaseQueue;


void LeggiElementoQueue(tipobaseQueue * p){
     scanf("%f",p);
     FFLUSH;
}

void VisualizzaElementoQueue(tipobaseQueue x){
     printf("\n\nElemento = %f \n\n", x);
}
```

Nel caso in cui il tipobaseQueue fosse uno struct composto da un cognome, nome e età:

```
#define S 15


typedef struct {
        char cognome[S], nome[S];
        unsigned int eta;
}tipobaseQueue;
```

si dovrebbe aggiungere al programma le seguenti funzioni:

```
void LeggiStringa(char s[],unsigned long dim){
  unsigned long i=0;

  for (i=0; i<dim-1;i++)
       if ((s[i]=getchar())=='\n') break;
  if (i==dim-1) FFLUSH;
  s[i]='\0';
}
```

Struttura Dati Coda

```
void LeggiElementoQueue(tipobaseQueue * p){
     printf("\nInserisci il Cognome ");
     LeggiStringa(p->cognome, S);
     printf("\nInserisci il Nome ");
     LeggiStringa(p->nome, S);
     printf("\nInserisci l'eta' ");
     scanf("%u",&p->eta);
     FFLUSH;
}

void VisualizzaElementoQueue(tipobaseQueue x){
     printf("\nCognome = %s ", x.cognome);
     printf("\nNome = %s ", x.nome);
     printf("\nEta' = %u \n\n",x.eta);
}
```

Nel seguito viene mostrato l'esempio di programma. Si noti che il programma potrebbe essere applicato sia all'implementazione con puntatori sia con vettori. Lo studente provi il seguente programma considerando entrambe le implementazioni mostrate in questo documento.

```
queue c;
tipobaseQueue elem;
unsigned short s;

int main(void)
{
 MakeNullQueue(&c);

 do {
   printf ("\n\nMenu di Operazioni");
   printf ("\n1-Inserimento");
   printf ("\n2-Cancellazione ");
   printf ("\n3-Svuota Coda ");
   printf ("\n4-Ispeziona Elemento in Testa ");
   printf ("\n5-Fine ");
   printf ("\nInserisci la scelta ");
   scanf ("%hu", &s);
   FFLUSH;

   switch(s){

     case 1 : if (!FullQueue(c)) {
                   printf("\nInserisci Elemento ");
                   LeggiElementoQueue(&elem);
                   EnQueue(&c,elem);
              } else printf ("\nCoda Piena \n");
              break;

     case 2 : if (EmptyQueue(c)) printf ( "\nCoda Vuota\n");
              else {
                   DeQueue(&c);
                   printf ( "\nPrimo Elemento Estratto\n");
              }
              break;

     case 3 : while (!EmptyQueue(c)) DeQueue(&c);
              printf ( "\nCoda Vuota\n");
```

```
              break;

     case 4 : if (EmptyQueue(c)) printf ( "\nCoda Vuota\n");
              else {
                   printf("\nVisualizza Elemento Testa ");
                   elem=Front(c);
                   VisualizzaElementoQueue(elem);
              }
              break;
   }
 } while (s<5);
}
```