Esempio di gestione di una lista

Nel seguito si codifica l'esempio della funzioni relative ad una lista

```
#include <stdio.h>
#include <alloc.h>

#define NULL 0

typedef struct node  /* elemento della lista */
    {   int item;
        struct node *next;
    } NODE;

NODE *first, *last;
/*puntatori agli elementi iniziale e finale della lista*/

void Create ()   /* inizializza la lista */
{ first = NULL;  last = NULL; }

void End ()      /* riazzera la lista */
{   int i;
while (first != NULL) i = DequeueF();
}


int  IsIn (int i)
/* verifica la presenza di un elemento nella lista */
{   NODE *t;
t = first;
while (t != NULL && t ->item != i) t = t ->next;
return(t != NULL);
}
```

```
int  Empty ()    /* verifica se la lista e' vuota o meno
*/
{   return (first == NULL);  }


int  Length ()   /* numero degli elementi in lista */
{   int count = 0;
NODE *temp = first;
while (temp != NULL)
    { count++; temp = temp -> next;}
return (count);
}

void EnqueueF (int i)
/* aggiunge nella lista al primo posto (in testa) */
{ NODE *newnode;
newnode = (NODE *) malloc(sizeof(NODE));
newnode ->next = first;  newnode ->item = i;
if (first == NULL) { last = newnode;}
first = newnode;
}

void EnqueueL (int i)
/* aggiunge nella lista all'ultimo posto (in coda) */
newnode = (NODE *) malloc(sizeof(NODE));
newnode ->next = NULL;   newnode ->next = NULL;
newnode ->item = i;
if (first == NULL)
    { first = newnode;   last = newnode; }
else
    { last->next = newnode;  last = newnode; }
}
```

```
void Enqueue (int i)
/* aggiunge solo se non c'e' nella lista */
{   NODE *t = first;
while (t != NULL && t -> item != i)
    t = t -> next;
if (t == NULL) /* inserisci l'elemento */
EnqueueF (i);
}
static Dealloca (struct node *temp)
/* funzione PRIVATA  non visibile all'esterno del file*/
{ int reply;
reply = temp -> item; free((char *) temp);
return reply;
}

int DequeueF  ()
/* toglie il primo elemento dalla lista (dalla testa) */
{   NODE *temp = first;
if (first == NULL) /* coda vuota */ return (NULL);
else {  if (first == last) /* un solo elemento */
        { first  = NULL; last = NULL; }
    else first = temp -> next;
    return Dealloca(temp);
    }
}

int DequeueL ()
/* toglie l'ultimo elemento dalla lista (dalla coda) */
{   NODE *old, *new, *temp = last;
if (first == NULL)  /* lista vuota */ return (NULL);
else {  if (first==last) /* il primo el. e' da togliere
*/
        { last  = NULL; first = NULL; }
    else  /* si lascia almeno un elemento */
        { old = first; new = old -> next;
          while (new != last)
            { old = new; new = new -> next;}
          last = old;  old ->next = NULL; }
    return Dealloca(temp);   }
}
```

```
int Dequeue (int i)
{ if (first == NULL)  /* lista vuota */ return (NULL);
  else { if (first -> item == i)
            /* il primo elemento da togliere*/
            return DequeueF();
          else
            {   NODE *t, *temp;
          t = first; temp = t -> next;
          while (temp !=NULL && temp ->item != i)
            { t = temp; temp = t -> next; }
          if (temp != NULL)  /*l'elemento c'e' */
            { if (t -> next == last) last = t;
              t -> next = temp -> next;
              return Dealloca(temp);  }
          else return NULL; /* l'elemento non c'e' */ }
}


main ()
{printf ("inizio programma di prova
    della lista su piu' file\n");
Create();
printf("Elemento inserito con EnqueueF(12)\n");
EnqueueF(12);
printf("Elemento inserito con EnqueueL(24)\n");
EnqueueL(24);
printf("Elemento inserito con Enqueue(10)\n");
Enqueue(10);
printf("Elemento estratto da coda %d\n", DequeueL());
printf("Elemento estratto da testa %d\n",DequeueF());
printf("Elemento inserito con EnqueueF(120)\n");
EnqueueF(120);
printf("Elemento inserito con EnqueueL(294)\n");
EnqueueL(294);
printf("Elemento inserito con Enqueue(110)\n");
Enqueue(110);
printf("Elemento 120 estratto %d\n", Dequeue(120));
printf("Elemento estratto da testa %d\n",DequeueF());
printf("110 esiste? %d\n", IsIn(110));
printf("Lunghezza lista %d\n", Length());
End();
}
```

Esempio di applicazione sviluppata su più file: Una lista

Nel seguito si codifica l'esempio della funzioni relative ad una lista questa volta specificate su più file.

Primo file: INTERFACCIA di una LISTA
file *list.h* che i programmi per usare le liste devono includere cioé IMPORTARE

```
/* tutte le funzioni seguenti sono dichiarate esterne */
extern void Create (void), End (void), Enqueue(int);
extern void EnqueueF(int i), EnqueueL(int);
extern int DequeueF (void), DequeueL (void),
    Dequeue (int), IsIn (int),
    Empty (void), Length (void);
```

Secondo file: IMPLEMENTAZIONE di una LISTA
file *list.c* che contiene l'implementazione di tutte le funzioni di lista

```
#include <stdio.h>
#include <alloc.h>
#define NULL 0
typedef struct node  /* elemento della lista */
    {   int item;
        struct node *next;
    } NODE;
static NODE *first, *last;
/*puntatori agli elementi iniziale e finale della lista*/

void Create ()   /* inizializza la lista */
{ first = NULL;  last = NULL; }

void End ()      /* riazzera la lista */
{   int i;
while (first != NULL) i = DequeueF();
}
```

```
int  IsIn (int i)
/* verifica la presenza di un elemento nella lista */
{   NODE *t;
t = first;
while (t != NULL && t ->item != i) t = t ->next;
return(t != NULL);
}

int  Empty ()    /* verifica se la lista e' vuota o meno
*/
{   return (first == NULL);  }

int  Length ()   /* numero degli elementi in lista */
{   int count = 0;
NODE *temp = first;
while (temp != NULL)
    { count++; temp = temp -> next;}
return (count);
}

void EnqueueF (int i)
/* aggiunge nella lista al primo posto (in testa) */
{ NODE *newnode;
newnode = (NODE *) malloc(sizeof(NODE));
newnode ->next = first;  newnode ->item = i;
if (first == NULL) { last = newnode;}
first = newnode;
}

void EnqueueL (int i)
/* aggiunge nella lista all'ultimo posto (in coda) */
newnode = (NODE *) malloc(sizeof(NODE));
newnode ->next = NULL;   newnode ->next = NULL;
newnode ->item = i;
if (first == NULL)
    { first = newnode;   last = newnode; }
else
    { last->next = newnode;  last = newnode; }
}
```

```
void Enqueue (int i)
/* aggiunge solo se non c'e' nella lista */
{   NODE *t = first;
while (t != NULL && t -> item != i)
    t = t -> next;
if (t == NULL) /* inserisci l'elemento */
EnqueueF (i);
}
static Dealloca (struct node *temp)
/* funzione PRIVATA  non visibile all'esterno del file*/
{ int reply;
reply = temp -> item; free((char *) temp);
return reply;
}

int DequeueF  ()
/* toglie il primo elemento dalla lista (dalla testa) */
{   NODE *temp = first;
if (first == NULL) /* coda vuota */ return (NULL);
else {  if (first == last) /* un solo elemento */
        { first  = NULL; last = NULL; }
    else first = temp -> next;
    return Dealloca(temp);
    }
}

int DequeueL ()
/* toglie l'ultimo elemento dalla lista (dalla coda) */
{   NODE *old, *new, *temp = last;
if (first == NULL)  /* lista vuota */ return (NULL);
else {  if (first==last) /* il primo el. e' da togliere
*/
        { last  = NULL; first = NULL; }
    else  /* si lascia almeno un elemento */
        { old = first; new = old -> next;
          while (new != last)
             { old = new; new = new -> next;}
          last = old;  old ->next = NULL; }
    return Dealloca(temp);   }
}
```

```
int Dequeue (int i)
{ if (first == NULL)  /* lista vuota */ return (NULL);
  else { if (first -> item == i)
            /* il primo elemento da togliere*/
            return DequeueF();
           else
            {   NODE *t, *temp;
          t = first; temp = t -> next;
          while (temp !=NULL && temp ->item != i)
            { t = temp; temp = t -> next; }
          if (temp != NULL)  /*l'elemento c'e' */
            { if (t -> next == last) last = t;
              t -> next = temp -> next;
              return Dealloca(temp);  }
          else return NULL; /* l'elemento non c'e' */ }
}
```

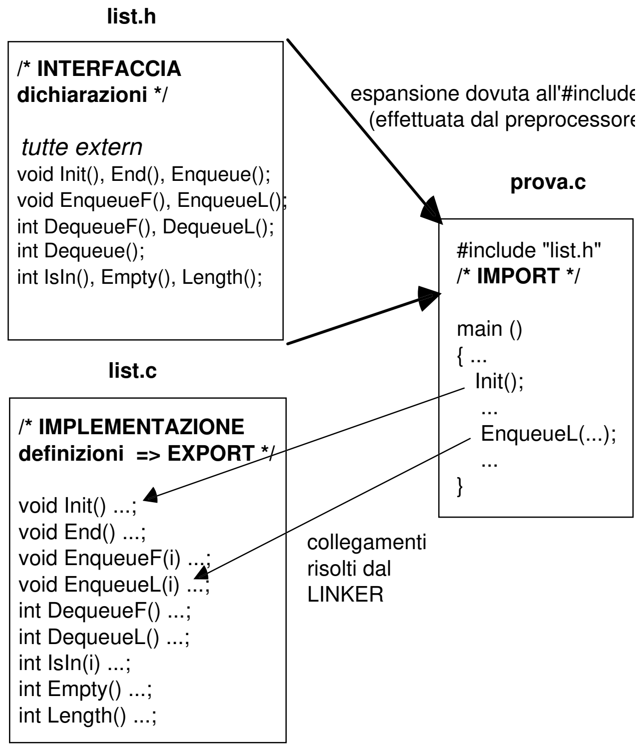
ESEMPIO di PROGRAMMA che usa la LISTA:

file *prova.c*:

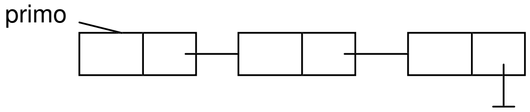
```
#include "list.h"
#include <stdio.h>

main ()
{printf ("inizio programma di prova
    della lista su piu' file\n");
Create();
printf("Elemento inserito con EnqueueF(12)\n");
EnqueueF(12);
printf("Elemento inserito con EnqueueL(24)\n");
EnqueueL(24);
printf("Elemento inserito con Enqueue(10)\n");
Enqueue(10);
printf("Elemento estratto da coda %d\n", DequeueL());
printf("Elemento estratto da testa %d\n",DequeueF());
printf("Elemento inserito con EnqueueF(120)\n");
EnqueueF(120);
printf("Elemento inserito con EnqueueL(294)\n");
EnqueueL(294);
printf("Elemento inserito con Enqueue(110)\n");
Enqueue(110);
printf("Elemento 120 estratto %d\n", Dequeue(120));
printf("Elemento estratto da testa %d\n",DequeueF());
printf("110 esiste? %d\n", IsIn(110));
printf("Lunghezza lista %d\n", Length());
End();
}
```

Per creare un UNICO ESEGUIBILE ===>
bisogna prima COMPILARE INDIPENDENTEMENTE i file *prova.c* e *list.c* e poi COLLEGARE insieme i file *oggetto corrispondenti prova.obj* e *list.obj*



ESERCIZIO: Gestione ricorsiva di Liste



E' un modo diverso di concepire e operare su una lista.
La lista è rappresentata dal puntatore al primo elemento

Le operazioni (funzioni) di base che si considerano sono (derivate dal linguaggio LISP):

Cons(elem, lista) ⇨ funzione di costruzione: inserisce l'elemento in testa alla lista
Head(lista) ⇨ ritorna il valore del primo elemento della lista
Tail(lista) ⇨ ritorna la lista ottenuta eliminando il primo elemento

Sulla base di queste funzioni base si possono costruire tutta una serie di altre funzioni, come:
writelista(lista): visualizza il contenuto della lista
Length(lista): restituisce il numero degli elementi componenti la lista
Member(E, lista): verifica se l'elemento E è presente nella lista
Append(E, lista): inserisce l'elemento dato E in coda alla lista (due versioni: nella prima versione **uso di semantica per copia**)

Se lista è una lista di interi:
Sum(lista): somma tutti gli elementi della lista

Se lista è una lista di elementi che possono essere ordinati:
Inserisci(E, lista): inserisce l'elemento dato E in ordine all'interno della lista

segue ESEMPIO
NOTA BENE: Solo nelle funzioni CONS, HEAD e TAIL ci si deve "sporcare" con i puntatori

```
#include <stdio.h>
#include <alloc.h>
#define nil 0

typedef enum { f, t } boolean;
typedef struct ELEMENTO
    {   int ITEM;
        struct ELEMENTO *NEXT;
    } ELEMENTO;

ELEMENTO * CONS (int elem, ELEMENTO *radice)
/*  funzione  di  costruzione  della  lista:  si  aggiunge
sempre  in  testa  ad  una  lista  preesistente  (all'inizio
sara' la lista nulla) */
{ ELEMENTO *Punt;
/* NOTA BENE: unico punto in cui si fa uso di malloc() */
Punt = (ELEMENTO *) malloc (sizeof(ELEMENTO));
Punt -> ITEM = elem;
Punt -> NEXT = radice;
return Punt;
};   /*CONS*/

int HEAD (ELEMENTO *radice)
/* funzione che ritorna il primo elemento della lista (la
testa della lista) */
{ if (radice == nil) return nil;
else  return radice -> ITEM;
};  /*HEAD*/

ELEMENTO * TAIL (ELEMENTO *radice)
/*   funzione   che   ritorna   la   lista   ottenuta   non
considerando il primo elemento (la coda della lista):
NOTA BENE: non si modifica la lista originale (non c'e'
side-effect) */
{ if (radice == nil) return (ELEMENTO *) nil;
else   return radice -> NEXT;
};  /*TAIL*/
```

```
void writelista (ELEMENTO *radice)
/* funzione che visualizza il contenuto della lista */
{ if (radice != nil)
    {   printf ("%d ", HEAD(radice));
        writelista(TAIL(radice)); }
else    { printf ("\n"); }
};  /*writelista*/
```

```
int LENGTH (ELEMENTO *radice)
/*  funzione  che  restituisce  il  numero  di  elementi
contenuti nella lista */
{ if (radice == nil) return 0;
else return (1 + LENGTH(TAIL(radice)));
};  /*LENGTH*/
```

```
boolean MEMBER (int el, ELEMENTO *radice)
/*  funzione che verifica l'esistenza dell'elemento dato
nella lista */
{ if (radice != nil)
    { if ( el == HEAD(radice) ) return t;
      else return ( MEMBER(el,TAIL(radice)) );
    };
/* else */
return f;
};  /*MEMBER*/
```

```
ELEMENTO *APPEND (int el, ELEMENTO *radice)
/* inserimento alla fine della lista: si costruisce una
nuova lista con un elemento in piu' in coda */
{ if (radice == nil)
    return (CONS(el, radice));
else return (CONS(HEAD(radice),APPEND(el,TAIL(radice))));
};  /*APPEND*/
```

```
void APPENDA (int el, ELEMENTO **pradice)
/* versione alternativa di APPEND ==> con side-effect:
non si costruisce una nuova lista, ma si appende alla
lista esistente */
{ if (*pradice == nil)
    *pradice = CONS(el, *pradice);
  else  APPENDA (el, &((*pradice) -> NEXT));
};  /*APPENDA*/
```

```
int SUM (ELEMENTO *radice)
/* lista di interi: somma tutti gli elementi */
{ if (radice == nil) return 0;
  else return ( HEAD(radice) + SUM(TAIL(radice)));
};  /*SUM*/
```

```
ELEMENTO * INSERISCI (int el, ELEMENTO *radice)
{ if (radice == nil)
    return (CONS(el, radice));
  else if (MEMBER (el, radice) != t)
        if (HEAD(radice) > el)
            return (CONS (el, radice));
            /*inserimento in testa*/
        else
  return (CONS(HEAD(radice),INSERISCI(el,TAIL(radice))));
    /* else se c'e' gia' non si inserisce*/
};  /*INSERISCI*/
```

```
main ()
{   ELEMENTO *ROOT = nil, *ROOT1;
    int CHOICE,EL,L;
    boolean FINE = f, R;

while (FINE != t)
{ printf ("\n\nQuale funzione vuoi eseguire sulla lista
        di interi ?\n");
printf ("1. CONS    2. HEAD   3. TAIL    4. LENGTH \n");
printf ("5. SUM     6. MEMBER 7. APPEND  8. INSERT \n");
printf ("9. APPENDA 10. SHOW 11. ESCI       \n");
scanf ("%d", &CHOICE);

switch (CHOICE) {
case 1:
    printf ("Argomento da appendere in testa? ");
    scanf ("%d", &EL);
    ROOT = CONS(EL,ROOT);
    writelista(ROOT);
    break;
case 2:
    EL = HEAD(ROOT);
    printf("L'elemento ottenuto e'  %d\n", EL);
    break;
case 3:
    ROOT1 = TAIL(ROOT); writelista(ROOT1);
    break;
case 4:
    L = LENGTH(ROOT);
    printf ("La lunghezza della lista  e' %d\n", L);
    break;
case 5:
    printf ("Somma degli elementi %d\n", SUM(ROOT));
    break;
case 6:
    printf ("Qual e' l'argomento da ricercare? \n");
    scanf ("%d", &EL);
    R = MEMBER(EL,ROOT);
    if (R == t) printf ("YES\n");
    else printf ("FALSE\n");
    break;
```

```
case 7:
    printf ("L'argomento da appendere in coda?\n ");
    scanf ("%d", &EL);
    ROOT = APPEND(EL,ROOT);
    writelista(ROOT);
    break;
case 8:
    printf ("L'argomento da inserire in ordine? ");
    scanf ("%d", &EL);
    ROOT = INSERISCI(EL,ROOT);
    writelista(ROOT);
    break;
case 9:
    printf ("L'argomento da appendere in coda
        (con SIDE-EFFECTS)?\n");
    scanf ("%d", &EL);
    APPENDA(EL,&ROOT);
    writelista(ROOT);
    break;
case 10:
    writelista (ROOT);
    break;

case 11:
    FINE = t;
    break;
        }  /*case*/
}    /*while*/
}     /*main*/
```

LISTA GENERICA

L'idea e' quella di poter lavorare su una lista indipendentemente da quello che la lista contiete come dati → dati generici!!!!

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

#define NULL 0

/* definizione tipo booleano */
typedef enum {bFALSE,bTRUE} BOOLEAN;

/* Descrizione dell'elemento della coda generica */
typedef struct QueueEl
    {
        struct QueueEl *pqeNext;
        void   *item; /* puntatore a void come tipo
generico */
    } QueueEl;

/* Descrizione dell'elemento di testa di una coda*/
typedef struct
        {
            QueueEl  *pqeFirst; /* punta al primo */
            QueueEl  *pqeLast; /* punta all'ultimo */
            int      cont;
        } HeadQueue;

/* marco che inizializza una coda */
#define INIT_QUEUE(phq) \
{\
    (phq)=(HeadQueue*)malloc(sizeof(HeadQueue));\
    (phq)->pqeFirst=NULL;\
    (phq)->pqeLast=NULL;\
    (phq)->cont=0;\
}
```

```
/* inserisce un elemento in coda */
int EnQueue (phq, Item)
HeadQueue  *phq;           /* Puntatore alla testa della
coda */
void   *Item;           /* Puntore all'elemento da
accodare */
{  int cont;
    QueueEl *pqe;

    pqe = (QueueEl *)malloc(sizeof(QueueEl));

    pqe->pqeNext = NULL;
    pqe->item = Item;

    if (phq->cont==0) /* se la lista e' vuota */
    {
     phq->pqeFirst=pqe;
     phq->pqeLast=pqe;
        }
      else /* se la lista non e' vuota */
    {
     phq->pqeLast->pqeNext=pqe;
     phq->pqeLast=pqe;
        }
    phq->cont=phq->cont+1;

    return(phq->cont);
} /* End EnQueue */
```

```
/*
toglie un elemento dalla coda sulla base
    di un pattern di ricerca
    di una funzione BOOLEAN che ricerca analizza tale
pattern
*/
void *DeQueue(phq, pat, compf)
HeadQueue  *phq;  /* Puntatore alla coda da cui si vuole
estrarre */
void       *pat;      /* pattern da contare */
BOOLEAN (*compf)();
{void *Item;
  QueueEl *pqePrec,*pqeNow;

  pqeNow=phq->pqeFirst;
  while (pqeNow!=NULL)
    {
     if(compf(pqeNow->item,pat)) /* se lo trovo esco dal
ciclo */
                     break;
     pqePrec=pqeNow; /* si tiene memorizzato il precedente
*/
     pqeNow=pqeNow->pqeNext; /* si passa al prossimo */
        }

  if (pqeNow!=NULL)   /* elemento trovato prima di
arrivare in fondo*/
      {
       if (pqeNow==phq->pqeFirst)
           phq->pqeFirst=pqeNow->pqeNext;  /* il secondo
diventa primo */
           /* eventualmente NULL se ce n'era solo uno */

           else
           pqePrec->pqeNext=pqeNow->pqeNext; /* il
precedente va al succ */

           if (pqeNow==phq->pqeLast) /* se il trovato e'
l'ultimo */
        if (phq->cont>1)
```

```
            phq->pqeLast=pqePrec; /* il penultimo diventa
ultimo */
                else
            phq->pqeLast=NULL; /* la lista e' vuota */

           phq->cont=phq->cont-1;
          }
  Item = (void *)(pqeNow->item);
  free(pqeNow);

return Item;
} /* End Dequeue */
```

```c
/* Banale: numero di elementi in coda */
int Lenght(phq)
HeadQueue  *phq;
{
return phq->cont;
}

/* Conta gli elementi corrispondenti a  un certo pattern
*/
int CountIn(pqeQ, pat, compf)
HeadQueue  *pqeQ;   /* Puntatore alla coda da cui si vuole
contare */
void       *pat;      /* pattern da contare */
BOOLEAN (*compf)();
{QueueEl *pqe;
  int contatore;

  contatore=0;
  pqe=pqeQ->pqeFirst;
  while(pqe!=NULL) /* !=NULL */
    { if (compf(pqe->item,pat)==bTRUE)
     {
       contatore++;
         }
      pqe=pqe->pqeNext;
     }

  return contatore;
} /* End CountIn */
```

```c
void *IsIn(phq, pat, compf)
HeadQueue    *phq;
void       *pat;
BOOLEAN    (*compf)();
{QueueEl *pqe;

  pqe=phq->pqeFirst;

  while(pqe)
     if (compf(pqe->item,pat)==bTRUE)
        return((void *)(pqe->item));
          else
        pqe=pqe->pqeNext;
     return(NULL);
}
```

```c
/* definizioni di vari tipi da aggiungere alla lista */
typedef struct {
        int numero;
               } Number;

BOOLEAN NumCmp(nn,num)
Number *nn;
int *num;
{
return(BOOLEAN)(nn->numero == *num);
}

BOOLEAN StrCmp(s1,s2)
char *s1,*s2;
{
if(strcmp(s1,s2)==0)
   return bTRUE;
else
   return bFALSE;
}
```

```c
main(argc,argv)
int argc;
char **argv;
{
HeadQueue *Coda,*CodaS;
BOOLEAN Continue=bTRUE;
char scelta;

INIT_QUEUE(Coda);
INIT_QUEUE(CodaS);

while(Continue)
   {
    printf("\n\n\nOPZIONI\n");
    printf("\ta = aggiungi\n");
    printf("\te = elimina\n");
    printf("\tl = lunghezza\n");
    printf("\tc = conta\n");
    printf("\tr = ricerca\n");
    printf("\n\n\taggiungi stringa = s\n");
    printf("\n\n\tcerca    stringa = u\n");
    printf("\tf=fine\n");
    printf("Cosa Vuoi Fare? :"); scanf("%c", &scelta);

    switch(scelta)
      {
      case 'a':
          {
           Number *nn;
           nn=(Number *)malloc(sizeof(Number));
           printf("aggiungo\n");
               printf("\n Numero? "); scanf("%d", &(nn-
>numero));
           EnQueue(Coda,nn);
               }
           break;
      case 'e':
          {
           Number *nn;
               int numero;
```

```
                printf("\n Numero? ");scanf("%d", &numero);
            nn = (Number *)DeQueue(Coda,&numero,NumCmp);
            printf("Ho estratto il numero %d\n\n\n",nn-
>numero);
            getchar();
            free(nn);
               }
           break;
       case 'r':
           {
            int numero;

            printf("\n Numero? ");scanf("%d", &numero);
            if(IsIn(Coda, &numero, NumCmp))
            printf("L'elemento %d e' in coda\n", numero);
                else
            printf("Non in coda \n\n\n");
               }
           break;
       case 'c':
           {
            int numero, conta;
            printf("\n Numero? ");scanf("%d", &(numero));
                conta = CountIn(Coda, &numero, NumCmp);
            printf(" %d elementi %d in coda\n", conta,
numero);
               }
           break;
       case 'l':
            printf("La Lunghezza %d\n", Lenght(Coda));
            getchar();
            break;

       case 's':
           {
            char *nome;

            nome = (char *)malloc(20);

            printf("\nStringa?"); scanf("%s", nome);
            EnQueue(CodaS, nome);
               }
```

```
           break;
       case 'u':
           {
            char *nome;
            nome = (char *)malloc(20);
            printf("\nStringa da cercare ? ");scanf("%s",
nome);

            if(IsIn(CodaS,nome,StrCmp))
            printf("L'elemento %s e' in coda\n", nome);
                else
            printf("Non in coda\n");
                getchar();
               }
           break;
       case 'f':
            printf("finisco\n");
            Continue = bFALSE;
       }
   }

printf("Fine Programma\n\n");
}
```