Università degli Studi di Napoli - Federico II

Facoltà di
Scienze Matematiche, Fisiche e Naturali

Corso di Laurea in Informatica

Laboratorio di Algoritmi e Strutture Dati
A.A. 2005 / 2006

Prof. Aniello Murano

PROGETTO
VIGILE URBANO

Realizzato da

Marco Sommella
50 / 482

Giovanni Di Cecca
50 / 887

Virginia Bellino
408 / 466

http://www.dicecca.net

Indice

SEZIONE 1:
SCHEMA GRAFICO

Grafico delle chiamate (coda_lista)

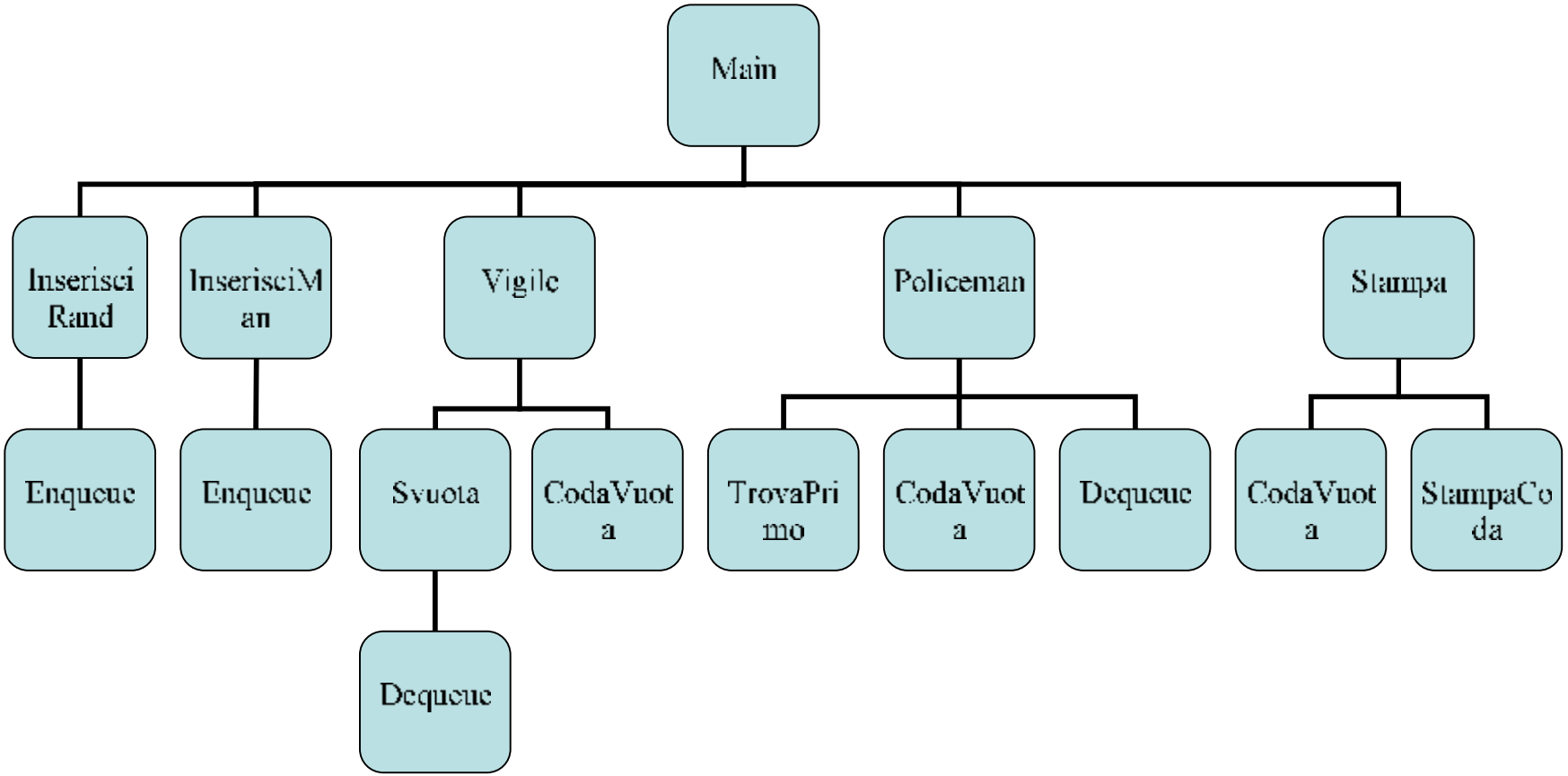
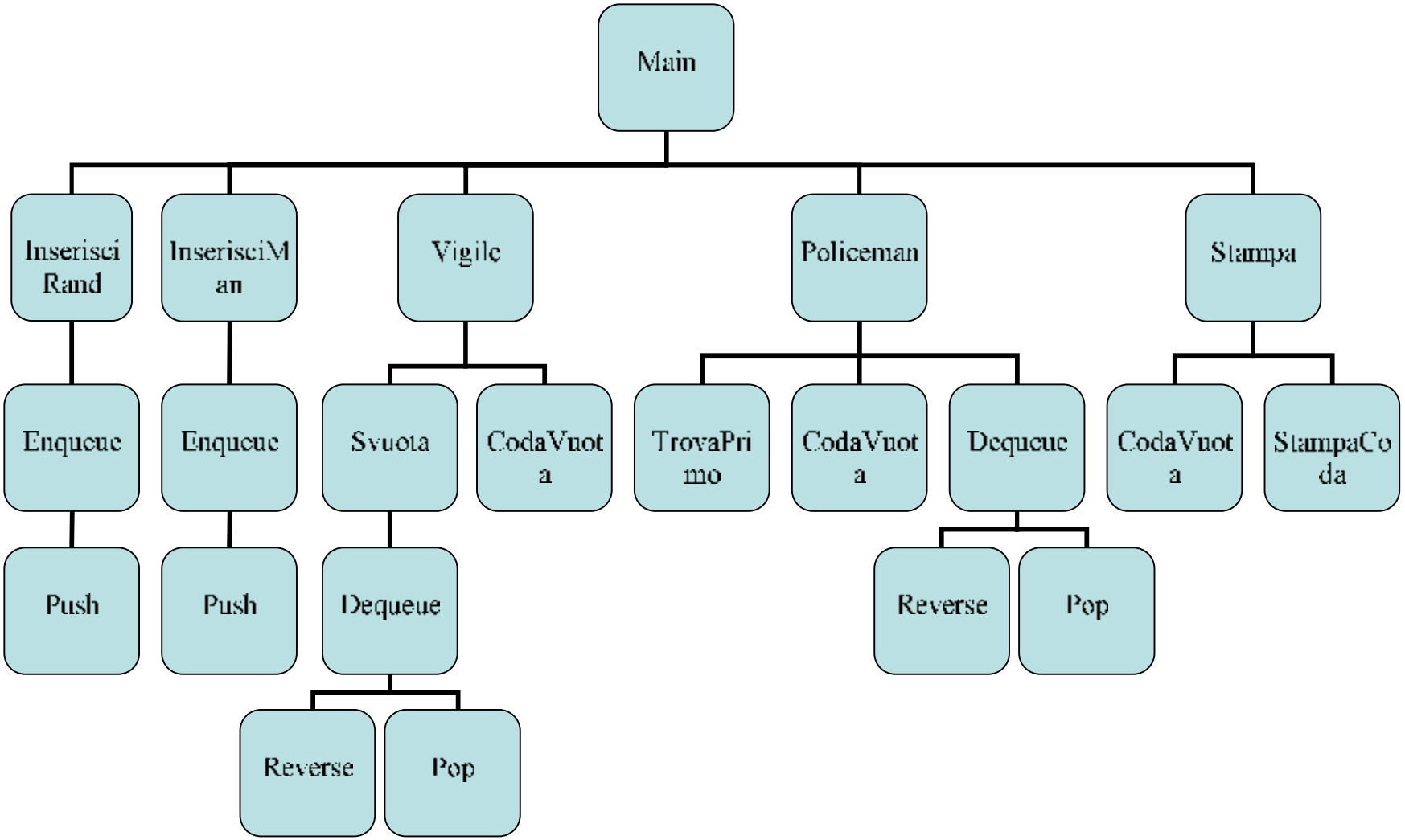


Grafico delle chiamate (coda_2stack)



SEZIONE 2:
DOCUMENTAZIONE ESTERNA

Documentazione esterna

- **Scopo:** il progetto si propone di realizzare la simulazione di un vigile urbano che regola il traffico in due tipologie di quadrivio, definite come segue:

  a) incrocio all'italiana
  in una delle quattro strade viene posto un segnale di precedenza, indicante che chi arriva in tale strada deve dare la precedenza a tutti. Nelle rimanenti 3 strade invece, coloro che arrivano danno precedenza a destra

  b) incrocio all'americana
  in questo caso, passa per primo chi arriva primo all'incrocio.

- **Specifiche:**

  void InserisciRand (CODA **nord, CODA **sud, CODA **est, CODA **ovest, int *ordine, int *prior, int *size);

  void InserisciMan (CODA **nord, CODA **sud, CODA **est, CODA **ovest, int *ordine, int *prior, int *size);

  void Stampa (CODA *nord, CODA *sud, CODA *est, CODA *ovest);

  void Svuota (CODA *coordinata, int size)

  void Vigile (CODA *nord, CODA *sud, CODA *est, CODA *ovest, int *ordine, int *size);

  int TrovaPrimo (int *ordine);

  void Policeman (CODA *nord, CODA *sud, CODA *est, CODA *ovest,int *ordine, int *prior, int *size);

Descrizione:

Aspetti generali

All'avvio del programma, l'utente, si trova di fronte ad un menu di selezione che prevede la possibilità di scegliere alcune opzioni.
Per quanto riguarda l'inserimento, vi sono due possibilità:

Inserimento randomizzato: il programma inserisce un numero di auto determinato dall'utente scegliendo casualmente in quale strada accodarle;

Inserimento manuale: l'utente sceglie quante auto inserire e dove accodarle.

In entrambi i casi, la targa che identifica l'auto è generata casualmente.
Per quanto riguarda la stampa, è possibile visualizzare la situazione dell'incrocio in ogni momento selezionando l'apposito comando dal menu di scelta.
Per far uscire le auto dall'incrocio, l'utente può selezionare due tipologie di vigile:

Italiano: una volta posizionato il segnale di precedenza in una delle quattro strade, il vigile svuota l'intero incrocio;

Americano: l'utente sceglie quante auto far uscire dall'incrocio e il vigile, seguendo la priorità di arrivo in testa alla coda, permette al suddetto numero di auto di lasciare l'incrocio.

Aspetti Implementativi

Abbiamo strutturato il programma su tre livelli:
una funzione main, che stampa il menu di scelta e richiama le funzioni;
un insieme di funzioni (incrocio), che materialmente gestisce le auto ed i vigili;
due librerie di funzioni interscambiabili che gestiscono le code.

Nella gestione delle code mediante due stack, poiché lo stack è stato simulato utilizzando un array dinamico e tale scelta rende impossibile eseguire direttamente l'eliminazione in coda, viene creato uno stack di appoggio in cui si riversano gli elementi per procedere successivamente alle eliminazione dalla testa di tale stack. Una volta che tale eliminazione è stata effettuata, gli elementi rimasti vengono nuovamente riversati nello stack originario ottenendo l'eliminazione richiesta.
Lo stack di appoggio viene creato ed eliminato all'interno della funzione dequeue; questa scelta è stata fatta per i seguenti motivi:
la stack di appoggio è funzionale alla sola eliminazione di un elemento in coda. Una volta che tale compito è stato assolto, la struttura non ha più ragione di esisteste. In questo modo si ottimizza l'utilizzo dello spazio di memoria, non influenzando la complessità asintotica.

Si noti che la presenza della variabile size tra i parametri di alcune funzioni di coda_lista è da attribuirsi alla necessità di avere lo stesso numero e tipo di parametri tra le funzioni di coda_lista e coda_2stack. All'interno di coda_lista size non viene mai utilizzata.

Nella gestione delle code mediante lista si è scelto di implementare la coda utilizzando una lista puntata circolare doppia con sentinella. E' stata scelta una

lista puntata circolare doppia poiché ciò consente di inserire in coda e eliminare in testa in un tempo costante [O(1)]. La presenza della sentinella invece, consente di non dover differenziare le istruzioni per effettuare l'inserimento in una coda vuota o in una con almeno un elemento. L'unico svantaggio legato all'uso della sentinella è rappresentato da un lieve spreco di spazio.

- **Indicazioni di utilizzo:**

il programma è stato compilato utilizzando il compilatore DevC++ versione 4.9.9.2.
Segnaliamo inoltre che, essendovi chiamate alla funzione system che comprendono comandi MS-DOS ("pause" et "cls"), se il programma viene compilato in ambiente Linux, il mancato riconoscimento di tali comandi da parte del sistema operativo comporta una visualizzazione dei risultati corretta ma poco ordinata.

- **Routine ausiliarie**

Il programma si avvale di numerose routine ausiliarie per la realizzazione delle scelte implementative da noi effettuate. Tali routine sono racchiuse in due librerie, denominate come segue:

a) coda_2stack
libreria per gestire una coda con 2 stack (uno fisso e uno di appoggio)

```
int CodaVuota ( CODA *targa );

CODA *CreaCoda ( int size );

void Push ( CODA *targa, int valore );

int Pop ( CODA *targa );

void StampaCoda ( CODA *targa );

void Reverse ( CODA *targa, CODA *appoggio );

CODA *Enqueue ( CODA *targa, int valore, int *size );

int Dequeue ( CODA *targa, int size );

typedef int CODA;
```

b) coda_lista

libreria per gestire una coda come lista puntata circolare doppia con sentinella

```
CODA *CreaCoda ( int size );

int CodaVuota ( CODA *testa );

CODA *Enqueue ( CODA *testa, int intero, int *size );

int Dequeue ( CODA *testa, int size );

void StampaCoda ( CODA *testa );
```

- **Complessità computazionale**

complessità di tempo
in entrambe le implementazioni adottate nel progetto, la complessità di tempo è O(n).

complessità di spazio
in entrambe le implementazioni adottate nel progetto, la complessità di spazio è O(n).

Esempio d'uso

```
************************************************************
* SIMULAZIONE INCROCIO ALL'ITALIANA E ALL'AMERICANA *
* REALIZZATO DA:                                           *
* Marco Sommella 50/482                                    *
* Giovanni Di Cecca 50/887                                 *
* Virginia Bellino 408/466                                 *
************************************************************

r: Inserisci Random
m: Inserisci Manuale
i: Vigile Italiano
a: Vigile Americano
s: Stampa Situazione
q: Quit
Scegli: r
```

Inserimento random di 8 auto:

```
Quante auto si desidera inserire: 8

Inserimento Random effettuato.
Stampa dal menu per avere la situazione dell'incrocio
Premere un tasto per continuare . . .
```

Inserimento manuale di ulteriori 2 auto:

```
Quante auto si desidera inserire: 2
Dove devo inserire l'auto targata 5727 ?
0 - NORD
1 - EST
2 - SUD
3 - OVEST
Scegli: 3
Dove devo inserire l'auto targata 11499 ?
0 - NORD
1 - EST
2 - SUD
3 - OVEST
Scegli: 0

Inserimento Manuale terminato.
Stampa dal menu per avere la situazione dell'incrocio
Premere un tasto per continuare . . .
```

Stampa:

```
A NORD ci sono in coda le auto targate:
19491   22795   2283    11499

A EST ci sono in coda le auto targate:
25056   5621

A SUD ci sono in coda le auto targate:
17535

A OVEST ci sono in coda le auto targate:
25102   5404    5727

Premere un tasto per continuare . . .
```

Vigile Americano:

```
Quante auto devo far uscire dall'incrocio: 5

L'auto targata 25056 ha lasciato la strada EST

L'auto targata 19491 ha lasciato la strada NORD

L'auto targata 25102 ha lasciato la strada OVEST

L'auto targata 17535 ha lasciato la strada SUD

L'auto targata 5621 ha lasciato la strada EST
Premere un tasto per continuare . . .
```

Vigile Italiano:

```
Dove devo inserire il segnale di precedenza ?
0 - NORD
1 - EST
2 - SUD
3 - OVEST
Scegli: 2

Auto che lasciano la strada OVEST:
5404
5727

Auto che lasciano la strada NORD:
22795
2283
11499

A EST non ci sono auto in coda

A SUD non ci sono auto in coda
Premere un tasto per continuare . . .
```

Stampa:

```
A NORD non ci sono auto in coda

A EST non ci sono auto in coda

A SUD non ci sono auto in coda

A OVEST non ci sono auto in coda

Premere un tasto per continuare . . .
```

SEZIONE 3:
CODICE C DEL PROGETTO

```c
/*                          MAIN.C
REALIZZATO DA:
Marco Sommella 50/482
Giovanni Di Cecca 50/887
Virginia Bellino 408/466
*/

#include <stdio.h>

/* il programma main utilizza, a scelta del programmatore, una delle 2
  librerie ausiliarie sottostanti; tale situazione è evidenziata
  commentando l'include che non si desidera utilizzare */

/* #include "coda_lista.c" */
#include "coda_2stack.c"

#include "incrocio.c"



main()
{
  char scelta; /* per il menu */
  int *ordine, /* puntatore ad array di 4 elementi contenente la priorità
                  attuale delle strade */
      prior, /* priorità nell'incrocio all'americana */
      *size; /* puntatore ad array di 4 elementi contenente la dim
                dell'array che costituisce lo stack di ogni strada
                ( necessario a coda_2stack ) */


                /* INIZIALIZZAZIONE */
  if ( !( ordine = ( int * ) calloc ( 4, sizeof( int ) ) ) )
  {
    printf ( "\nErrore di allocazione, terminare il programma\n" );
    system ( "PAUSE" );
  }
  if ( !( size = ( int * ) calloc ( 4, sizeof( int ) ) ) )
  {
    printf ( "\nErrore di allocazione, terminare il programma\n" );
    system ( "PAUSE" );
  }
  prior = 0;
  CODA* nord = CreaCoda( size[0] );
  CODA* est = CreaCoda( size[1] );
  CODA* sud = CreaCoda( size[2] );
  CODA* ovest = CreaCoda( size[3] );
  srand ( time ( NULL ) ); /* imposta la scelta del seme usando il clock */


                /* MENU DI SELEZIONE */
  do
  {
    system ( "CLS" );
    printf ( "*****************************************************\n" );
    printf ( "* SIMULAZIONE INCROCIO ALL'ITALIANA E ALL'AMERICANA *\n" );
    printf ( "* REALIZZATO DA:                                    *\n" );
    printf ( "* Marco Sommella 50/482                             *\n" );
    printf ( "* Giovanni Di Cecca 50/887                         *\n" );
    printf ( "* Virginia Bellino 408/466                          *\n" );
```

```
61:     printf ( "*************************************************\n\n" );
62:
63:     printf( "r: Inserisci Random\nm: Inserisci Manuale\n" );
64:     printf( "i: Vigile Italiano\na: Vigile Americano\n" );
65:     printf( "s: Stampa Situazione\nq: Quit\nScegli: " );
66:     fflush( stdin );
67:     scanf( "%c", &scelta );
68:     switch( scelta ) /* l'utente può inserire lettere maiuscole o
69:                         minuscole indifferentemente */
70:     {
71:       case 'R':
72:       case 'r': InserisciRand ( &nord, &sud, &est, &ovest,
73:                                 ordine, &prior, size );
74:                 break;
75:       case 'M':
76:       case 'm': InserisciMan ( &nord, &sud, &est, &ovest,
77:                                 ordine, &prior, size );
78:                 break;
79:       case 'I':
80:       case 'i': Vigile ( nord, sud, est, ovest, ordine, size );
81:                 break;
82:       case 'A':
83:       case 'a': Policeman ( nord, sud, est, ovest, ordine, &prior, size );
84:                 break;
85:       case 'S':
86:       case 's': Stampa ( nord, sud, est, ovest );
87:                 break;
88:       default:  break;
89:     }
90:   }
91:   while ( scelta != 'q' && scelta != 'Q' );
92:   free ( ordine );
93:   free ( size );
94: }
95:
```

```c
/*                          INCROCIO.C
REALIZZATO DA:
Marco Sommella 50/482
Giovanni Di Cecca 50/887
Virginia Bellino 408/466
*/

#include "incrocio.h"

/* permette l'inserimento random di un numero a scelta di auto nelle strade

  INPUT:  **nord - puntatore a puntatore alla coda della strada a nord
          **sud - puntatore a puntatore alla coda della strada a sud
          **est - puntatore a puntatore alla coda della strada a est
          **ovest - puntatore a puntatore alla coda della strada a ovest
          *ordine - puntatore ad array contenente la priorità delle strade
          *prior - indice di priorità
          *size - puntatore ad array contenente la dimensione dell'array che
                  rappresenta lo stack ( necessario per coda_2stack )
  OUTPUT: **nord - puntatore a puntatore alla coda della strada a nord
          **sud - puntatore a puntatore alla coda della strada a sud
          **est - puntatore a puntatore alla coda della strada a est
          **ovest - puntatore a puntatore alla coda della strada a ovest
          *ordine - puntatore ad array contenente la priorità delle strade
          *prior - indice di priorità
          *size - puntatore ad array contenente la dimensione dell'array che
                  rappresenta lo stack ( necessario per coda_2stack )
*/
void InserisciRand ( CODA **nord, CODA **sud, CODA **est, CODA **ovest,
                     int *ordine, int *prior, int *size )
{
  int i, /* indice per il for */
      num, /* numero di auto da inserire */
      luogo, /* strada in cui vengono inserite le auto */
      targa; /* targa dell'auto inserita */
  system ( "CLS" );
  printf ( "Quante auto si desidera inserire: " );
  scanf ( "%d", &num );
  for ( i = 0; i < num; i++ )
  {
    luogo = rand() % 4; /* num auto inserite casualmente nelle 4 strade */
    targa = rand();
    switch ( luogo )
    {
      case 0: *nord = Enqueue ( *nord, targa, &size[0] );
              if ( ordine[0] == 0 ) /* se è stato inserito il primo elemento
                                       cioè se alla coda non è stata assegnata
                                       ancora nessuna priorità */
              {
                ordine[0] = ++*prior;
              }
              break;
      case 1: *est = Enqueue ( *est, targa, &size[1] );
              if ( ordine[1] == 0 )
              {
                ordine[1] = ++*prior;
              }
              break;
      case 2: *sud = Enqueue ( *sud, targa, &size[2] );
              if ( ordine[2] == 0 )
```

```
61:                {
62:                  ordine[2] = ++*prior;
63:                }
64:                break;
65:       case 3: *ovest = Enqueue ( *ovest, targa, &size[3] );
66:                if ( ordine[3] == 0 )
67:                {
68:                  ordine[3] = ++*prior;
69:                }
70:                break;
71:     }
72:   }
73:   printf ( "\nInserimento Random effettuato.\n" );
74:   printf ( "Stampa dal menu per avere la situazione dell'incrocio\n" );
75:   system ( "PAUSE" );
76: }
77:
78: /* permette l'inserimento manuale di un numero a scelta di auto nelle strade
79:
80:   INPUT:  **nord - puntatore a puntatore alla coda della strada a nord
81:           **sud - puntatore a puntatore alla coda della strada a sud
82:           **est - puntatore a puntatore alla coda della strada a est
83:           **ovest - puntatore a puntatore alla coda della strada a ovest
84:           *ordine - puntatore ad array contenente la priorità delle strade
85:           *prior - indice di priorità
86:           *size - puntatore ad array contenente la dimensione dell'array che
87:                   rappresenta lo stack ( necessario per coda_2stack )
88:   OUTPUT: **nord - puntatore a puntatore alla coda della strada a nord
89:           **sud - puntatore a puntatore alla coda della strada a sud
90:           **est - puntatore a puntatore alla coda della strada a est
91:           **ovest - puntatore a puntatore alla coda della strada a ovest
92:           *ordine - puntatore ad array contenente la priorità delle strade
93:           *prior - indice di priorità
94:           *size - puntatore ad array contenente la dimensione dell'array che
95:                   rappresenta lo stack ( necessario per coda_2stack )
96: */
97: void InserisciMan ( CODA **nord, CODA **sud, CODA **est, CODA **ovest,
98:                int *ordine, int *prior, int *size )
99: {
100:   int i, /* indice per il for */
101:       num, /* numero di auto da inserire */
102:       luogo, /* strada in cui vengono inserite le auto */
103:       targa; /* targa dell'auto inserita */
104:   system ( "CLS" );
105:   printf ( "Quante auto si desidera inserire: " );
106:   scanf( "%d", &num );
107:   for ( i = 0; i < num; i++ )
108:   {
109:     targa = rand();
110:     printf ( "Dove devo inserire l'auto targata %d ?\n", targa );
111:     printf ( "0 - NORD\n1 - EST\n2 - SUD\n3 - OVEST\nScegli: " );
112:     scanf( "%d", &luogo );
113:     switch ( luogo )
114:     {
115:       case 0: *nord = Enqueue ( *nord, targa, &size[0] );
116:                if ( ordine[0] == 0 ) /* se è stato inserito il primo elemento
117:                                         cioè se alla coda non è stata assegnata
118:                                         ancora nessuna priorità */
119:                {
120:                  ordine[0] = ++*prior;
```

```
121:                }
122:                break;
123:        case 1: *est = Enqueue ( *est, targa, &size[1] );
124:                if ( ordine[1] == 0 )
125:                {
126:                  ordine[1] = ++*prior;
127:                }
128:                break;
129:        case 2: *sud = Enqueue ( *sud, targa, &size[2] );
130:                if ( ordine[2] == 0 )
131:                {
132:                  ordine[2] = ++*prior;
133:                }
134:                break;
135:        case 3: *ovest = Enqueue ( *ovest, targa, &size[3] );
136:                if ( ordine[3] == 0 )
137:                {
138:                  ordine[3] = ++*prior;
139:                }
140:                break;
141:      }
142:    }
143:    printf ( "\nInserimento Manuale terminato.\n" );
144:    printf ( "Stampa dal menu per avere la situazione dell'incrocio\n" );
145:    system ( "PAUSE" );
146: }
147:
148: /* svuota una strada
149:
150:    INPUT:  *coordinata - puntatore a una coda
151:            size - intero contenente la dimensione dell'array che
152:                    rappresenta lo stack ( necessario per coda_2stack ) della
153:                    coda che si vuole svuotare
154:    OUTPUT: *coordinata - puntatore a una coda
155: */
156: void Svuota ( CODA *coordinata, int size )
157: {
158:    while ( !CodaVuota ( coordinata ) )
159:    {
160:      printf ( "%d\n", Dequeue ( coordinata, size ) );
161:    }
162: }
163:
164: /* permette di svuotare l'incroncio secondo le regole italiane
165:
166:    INPUT:  *nord - puntatore alla coda della strada a nord
167:            *sud - puntatore alla coda della strada a sud
168:            *est - puntatore alla coda della strada a est
169:            *ovest - puntatore alla coda della strada a ovest
170:            *ordine - puntatore ad array contenente la priorità delle strade
171:            *size - puntatore ad array contenente la dimensione dell'array che
172:                    rappresenta lo stack ( necessario per coda_2stack )
173:    OUTPUT: *nord - puntatore alla coda della strada a nord
174:            *sud - puntatore alla coda della strada a sud
175:            *est - puntatore alla coda della strada a est
176:            *ovest - puntatore alla coda della strada a ovest
177:            *ordine - puntatore ad array contenente la priorità delle strade
178: */
179: void Vigile ( CODA *nord, CODA *sud, CODA *est, CODA *ovest,
180:            int *ordine, int *size )
```

```
{
  int precedenza, /* indice della strada in cui viene posta la precedenza */
      i; /* indice per il for */
  system ( "CLS" );
  printf ( "Dove devo inserire il segnale di precedenza ?\n" );
  printf ( "0 - NORD\n1 - EST\n2 - SUD\n3 - OVEST\nScegli: " );
  scanf( "%d", &precedenza );
  for ( i = 1; i <= 4; i++ )
  {
    switch ( ( precedenza + i ) % 4 ) /* sfrutta la disposizione in senso
                                         orario dei numeri associati alle strade */
    {
      case 0: if ( CodaVuota ( nord ) )
              {
                printf ( "\nA NORD non ci sono auto in coda\n" );
              }
              else
              {
                printf ( "\nAuto che lasciano la strada NORD:\n" );
                Svuota ( nord, size[0] );
              }
              break;
      case 1: if ( CodaVuota ( est ) )
              {
                printf ( "\nA EST non ci sono auto in coda\n" );
              }
              else
              {
                printf ( "\nAuto che lasciano la strada EST:\n" );
                Svuota ( est, size[1] );
              }
              break;
      case 2: if ( CodaVuota ( sud ) )
              {
                printf ( "\nA SUD non ci sono auto in coda\n" );
              }
              else
              {
                printf ( "\nAuto che lasciano la strada SUD:\n" );
                Svuota ( sud, size[2] );
              }
              break;
      case 3: if ( CodaVuota ( ovest ) )
              {
                  printf ( "\nA OVEST non ci sono auto in coda\n" );
              }
              else
              {
                printf ( "\nAuto che lasciano la strada OVEST:\n" );
                Svuota ( ovest, size[3] );
              }
              break;
    }
    ordine[i-1] = 0; /* assicura che tutte le code, poichè vuote,
                        non abbiamo una priorità assegnata */
  }
  system ( "PAUSE" );
}

/* ritorna la posizione del min ( maggiore di 0 )
```

```
241: in "ordine" oppure -1 se tutte le priorità sono a 0
242:
243:   INPUT:  *ordine - puntatore ad array contenente la priorità delle strade
244:   OUTPUT: ritorna la posizione del min ( maggiore di 0 )
245:           oppure -1 se tutte le priorità sono a 0
246: */
247: int TrovaPrimo ( int *ordine )
248: {
249:   int i, /* indice per il for */
250:       temp = 0, /* valore min durante la ricerca */
251:       trovato = -1; /* indice del valore min */
252:
253:   for ( i = 0; i < 4; i++ )
254:   {
255:     if ( ordine[i] > 0 && ( ordine[i] < temp | temp == 0 ) )
256:     {
257:       temp = ordine[i];
258:       trovato = i;
259:     }
260:   }
261:   return trovato;
262: }
263:
264: /* permette di far uscire dall'incroncio un numero a scelta di auto
265:     secondo le regole americane
266:
267:   INPUT:  *nord - puntatore alla coda della strada a nord
268:           *sud - puntatore alla coda della strada a sud
269:           *est - puntatore alla coda della strada a est
270:           *ovest - puntatore alla coda della strada a ovest
271:           *ordine - puntatore ad array contenente la priorità delle strade
272:           *prior - indice di priorità
273:           *size - puntatore ad array contenente la dimensione dell'array che
274:                   rappresenta lo stack ( necessario per coda_2stack )
275:   OUTPUT: *nord - puntatore alla coda della strada a nord
276:           *sud - puntatore alla coda della strada a sud
277:           *est - puntatore alla coda della strada a est
278:           *ovest - puntatore alla coda della strada a ovest
279:           *ordine - puntatore ad array contenente la priorità delle strade
280:           *prior - indice di priorità
281: */
282: void Policeman ( CODA *nord, CODA *sud, CODA *est, CODA *ovest,
283:                  int *ordine, int *prior, int *size )
284: {
285:   int num, /* numero di auto che deve lasciare l'incrocio */
286:       i; /* indice per il for */
287:   system ( "CLS" );
288:   printf ( "Quante auto devo far uscire dall'incrocio: " );
289:   scanf( "%d", &num );
290:   for ( i = 0; i < num; i++ )
291:   {
292:     switch ( TrovaPrimo( ordine ) )
293:     {
294:       case 0: printf ( "\nL'auto targata %d ha lasciato la strada NORD\n",
295:                        Dequeue( nord, size[0] ) );
296:               if ( CodaVuota ( nord ) )
297:               {
298:                 ordine[0] = 0; /* in questo modo la funzione TrovaPrimo
299:                                   non considererà più questa coda come un
300:                                   candidato possibile nella sua ricerca */
```

```
301:                }
302:                else
303:                {
304:                ordine[0] = ++*prior; /* assegna una nuova priorità
305:                                         maggiore di tutte le altre */
306:                }
307:                break;
308:        case 1: printf ( "\nL'auto targata %d ha lasciato la strada EST\n",
309:                          Dequeue( est, size[1] ) );
310:                if ( CodaVuota( est ) )
311:                {
312:                  ordine[1] = 0;
313:                }
314:                else
315:                {
316:                  ordine[1] = ++*prior;
317:                }
318:                break;
319:        case 2: printf ( "\nL'auto targata %d ha lasciato la strada SUD\n",
320:                          Dequeue( sud, size[2] ) );
321:                if ( CodaVuota( sud ) )
322:                {
323:                  ordine[2] = 0;
324:                }
325:                else
326:                {
327:                  ordine[2] = ++*prior;
328:                }
329:                break;
330:        case 3: printf ( "\nL'auto targata %d ha lasciato la strada OVEST\n",
331:                          Dequeue( ovest, size[3] ) );
332:                if ( CodaVuota( ovest ) )
333:                {
334:                  ordine[3]=0;
335:                }
336:                else
337:                {
338:                  ordine[3]=++*prior;
339:                }
340:                break;
341:        case -1:  printf ( "\nNon ci sono piu' auto all'incrocio !!!\n" );
342:                  break;
343:      }
344:    }
345:    system ( "PAUSE" );
346: }
347:
348: /* stampa la situazione del incrocio
349:
350:    INPUT:  *nord - puntatore alla coda della strada a nord
351:            *sud - puntatore alla coda della strada a sud
352:            *est - puntatore alla coda della strada a est
353:            *ovest - puntatore alla coda della strada a ovest
354:    OUTPUT:
355: */
356: void Stampa ( CODA *nord, CODA *sud, CODA *est, CODA *ovest )
357: {
358:    system ( "CLS" );
359:    if ( CodaVuota ( nord ) )
360:    {
```

```
361:     printf ( "\nA NORD non ci sono auto in coda\n" );
362:   }
363:   else
364:   {
365:     printf ( "\nA NORD ci sono in coda le auto targate:\n" );
366:     StampaCoda ( nord );
367:   }
368:   if ( CodaVuota ( est ) )
369:   {
370:     printf ( "\nA EST non ci sono auto in coda\n" );
371:   }
372:   else
373:   {
374:     printf ( "\nA EST ci sono in coda le auto targate:\n" );
375:     StampaCoda ( est );
376:   }
377:   if ( CodaVuota ( sud ) )
378:   {
379:     printf ( "\nA SUD non ci sono auto in coda\n" );
380:   }
381:   else
382:   {
383:     printf ( "\nA SUD ci sono in coda le auto targate:\n" );
384:     StampaCoda ( sud );
385:   }
386:   if ( CodaVuota ( ovest ) )
387:   {
388:     printf ( "\nA OVEST non ci sono auto in coda\n" );
389:   }
390:   else
391:   {
392:     printf ( "\nA OVEST ci sono in coda le auto targate:\n" );
393:     StampaCoda ( ovest );
394:   }
395:   printf ( "\n" );
396:   system ( "PAUSE" );
397: }
398:
```

```
 1: /*                          INCROCIO.H
 2: REALIZZATO DA:
 3: Marco Sommella 50/482
 4: Giovanni Di Cecca 50/887
 5: Virginia Bellino 408/466
 6: */
 7:
 8: /* prototipi */
 9:
10:
11: /* RESPONSABILE: Bellino */
12: void InserisciRand ( CODA **nord, CODA **sud, CODA **est, CODA **ovest,
13:                int *ordine, int *prior, int *size );
14:
15:
16: /* RESPONSABILE: Bellino */
17: void InserisciMan ( CODA **nord, CODA **sud, CODA **est, CODA **ovest,
18:                int *ordine, int *prior, int *size );
19:
20:
21: /* RESPONSABILE: Di Cecca */
22: void Svuota ( CODA *coordinata, int size );
23:
24:
25: /* RESPONSABILE: Di Cecca */
26: void Vigile ( CODA *nord, CODA *sud, CODA *est, CODA *ovest,
27:           int *ordine, int *size );
28:
29:
30: /* RESPONSABILE: Sommella */
31: int TrovaPrimo ( int *ordine );
32:
33:
34: /* RESPONSABILE: Sommella */
35: void Policeman ( CODA *nord, CODA *sud, CODA *est, CODA *ovest,
36:              int *ordine, int *prior, int *size );
37:
38:
39: void Stampa ( CODA *nord, CODA *sud, CODA *est, CODA *ovest );
40:
```

SEZIONE 4:
CODICE C DELLE ROUTINE AUSILIARIE

```c
/*                          CODA_2STACK.C
REALIZZATO DA:
Marco Sommella 50/482
Giovanni Di Cecca 50/887
Virginia Bellino 408/466
*/

/* libreria per gestire una coda come
  2 stack (uno fisso e uno di appoggio) */

#include <malloc.h>
#include "coda_2stack.h"

/* verifica se la coda è vuota ( ovvero se lo stack,
                                implementato come array, è vuoto )

  INPUT:  *targa - puntatore ad array che rappresenta uno stack
  OUTPUT: ritorna 0 se non vuota
                  1 se vuota
*/
int CodaVuota ( CODA *targa )
{
  return ( targa[0] == 0 ); /* poichè targa è un array che rappresenta lo stack
                               se la posizione 0 è uguale a 0
                               non ci sono elementi */
}

/* crea la coda ( ovvero lo stack implementato come array dinamico )

  INPUT:  size - dimensione iniziale dell'array che rappresenta lo stack
  OUTPUT: ritorna il puntatore all'array creato
*/
CODA *CreaCoda ( int size )
{
  CODA *tmp; /* array per contenere gli elementi dello stack */
  if ( !( tmp = (CODA *) calloc ( size + 1 , sizeof(CODA) ) ) )
  {
    printf ( "\nErrore di allocazione, terminare il programma\n" );
    system ( "PAUSE" );
  }
  return tmp;
}

/* permette l'inserimento di un elemento in uno stack implementato come array

  INPUT:  *targa - puntatore all'array che rappresenta lo stack
          valore - valore che sarà inserito
  OUTPUT:
*/
void Push ( CODA *targa, int valore )
{
  targa[++targa[0]] = valore;
}

/* permette l'eliminazione di un elemento da uno stack implementato come array

  INPUT:  *targa - puntatore all'array che rappresenta lo stack
  OUTPUT: ritorna il valore dell'elemento eliminato
*/
int Pop ( CODA *targa )
```

```
 61: {
 62:   return targa[targa[0]--];
 63: }
 64:
 65: /* riversa gli elementi di uno stack in un altro stack di appoggio
 66:
 67:   INPUT:  *targa - puntatore all'array che rappresenta lo stack
 68:           *appoggio - puntatore all'array di appoggio
 69:   OUTPUT:
 70: */
 71: void Reverse ( CODA *targa, CODA *appoggio )
 72: {
 73:   while ( !CodaVuota ( targa ) )
 74:     Push ( appoggio, Pop( targa ) );
 75: }
 76:
 77: /* permette l'inserimento di un elemento in una coda
 78:     gestita come stack ( implementato con array dinamici )
 79:
 80:   INPUT:  *targa - puntatore all'array che rappresenta lo stack
 81:           valore - valore che sarà inserito
 82:           *size - la dimensione dell'array che rappresenta lo stack
 83:   OUTPUT: *size - la nuova dimensione dell'array che rappresenta lo stack
 84:             ritorna il puntatore all'array che rappresenta lo stack
 85: */
 86: CODA *Enqueue ( CODA *targa, int valore, int *size )
 87: {
 88:   if ( !( targa[0] < *size ) ) /* se la spazio all'interno dello stack non è
 89:                                   sufficiente, quest'ultimo viene riallocato
 90:                                   ad una dimensione doppia della precendente */
 91:   {
 92:     if ( *size == 0 ) /* senza questo controllo se size=0 non aumenta mai */
 93:     {
 94:       *size = 1;
 95:     }
 96:     else
 97:     {
 98:       *size = ( *size * 2 );
 99:     }
100:
101:     targa = ( CODA * ) realloc ( targa, ( ( *size + 1 ) * sizeof( CODA ) ) );
102:   }
103:   Push ( targa, valore );
104:   return targa;
105: }
106:
107: /* permette l'eliminazione di un elemento da una coda
108:     gestita come stack ( implementato con array dinamici )
109:
110:   INPUT:  *targa - puntatore all'array che rappresenta lo stack
111:           *size - la dimensione dell'array che rappresenta lo stack
112:   OUTPUT: ritorna il valore dell'elemento eliminato
113: */
114: int Dequeue ( CODA *targa, int size )
115: {
116:   int temp; /* valore di ritorno */
117:   CODA *appoggio; /* crea uno stack di appoggio temporaneo,
118:                      necessario all'estrazione in coda dallo stack */
119:   if ( !( appoggio = ( CODA * ) calloc ( size + 1, sizeof( CODA ) ) ) )
120:   {
```

```
121:     printf ( "\nErrore di allocazione, terminare il programma\n" );
122:     system ( "PAUSE" );
123:   }
124:
125:   Reverse( targa, appoggio ); /* poichè non è possibile eliminare
126:                                  in coda da uno stack lo si riversa
127:                                  in uno d'appoggio*/
128:   temp = Pop( appoggio ); /* si elimina l'elemento */
129:   Reverse ( appoggio, targa ); /* e si ripristina lo stack di partenza
130:                                  ( meno l'elemento che ci interessava
131:                                    eliminare ) */
132:
133:   free ( appoggio );
134:   return temp;
135: }
136:
137: /* stampa gli elementi della coda ( ovvero gli elementi
138:                                     dello stack implementato come array )
139:
140:   INPUT:  *targa - puntatore all'array che rappresenta lo stack
141:   OUTPUT:
142: */
143: void StampaCoda ( CODA *targa )
144: {
145:   int i; /* indice per il for */
146:   for ( i = 1; i <= targa[0]; i++ )
147:   {
148:     printf ( "%d\t", targa[i] );
149:   }
150:   printf ( "\n" );
151: }
152:
```

```
 1: /*                          CODA_2STACK.H
 2: REALIZZATO DA:
 3: Marco Sommella 50/482
 4: Giovanni Di Cecca 50/887
 5: Virginia Bellino 408/466
 6:
 7: RESPONSABILI: Sommella - Di Cecca
 8: */
 9:
10: /* libreria per gestire una coda come
11:   2 stack (uno fisso e uno di appoggio) */
12:
13: typedef int CODA;
14:
15: /* prototipi */
16: int CodaVuota ( CODA *targa );
17:
18: CODA *CreaCoda ( int size );
19:
20: void Push ( CODA *targa, int valore );
21:
22: int Pop ( CODA *targa );
23:
24: void StampaCoda ( CODA *targa );
25:
26: void Reverse ( CODA *targa, CODA *appoggio );
27:
28: CODA *Enqueue ( CODA *targa, int valore, int *size );
29:
30: int Dequeue ( CODA *targa, int size );
31:
```

```c
/*                          CODA_LISTA.C
REALIZZATO DA:
Marco Sommella 50/482
Giovanni Di Cecca 50/887
Virginia Bellino 408/466
*/

/* libreria per gestire una coda come
  lista puntata circolare doppia con sentinella */

/* la presenza della variabile size tra i parametri di alcune funzioni è
  da attribuirsi alla necessità di avere lo stesso numero e tipo di parametri
  tra le funzioni di coda_lista e coda_2stack. All'interno di coda_lista size
  non viene mai utilizzata */

#include <malloc.h>
#include "coda_lista.h"

/* crea l'elemento sentinella che rappresenta la coda vuota

  INPUT:  size - not used
  OUTPUT: ritorna il puntatore al nodo sentinella
*/
CODA *CreaCoda ( int size )
{
  CODA *tmp; /* crea l'elemento sentinella */
  if ( !( tmp = ( CODA * ) malloc ( sizeof( CODA ) ) ) )
  {
    printf ( "\nErrore di allocazione, terminare il programma\n" );
    system ( "PAUSE" );
  }
  tmp->next = tmp;
  tmp->prev = tmp;
  return tmp;
}

/* verifica se la coda è vuota

  INPUT:  *testa - puntatore al nodo sentinella
  OUTPUT: ritorna 0 se non vuota
                  1 se vuota
*/
int CodaVuota ( CODA *testa )
{
  return ( testa == testa->next ); /* se il next della sentinella
                                      punta a se stesso la coda è vuota */
}

/* inserisce un elemento in coda ( alla fine della coda )

  INPUT:  *size - not used
          intero - numero da inserire
          *testa - puntatore al nodo sentinella
  OUTPUT: ritorna il puntatore al nodo sentinella
*/
CODA *Enqueue ( CODA *testa, int intero, int *size )
{
  CODA* tmp; /* un nuovo nodo da aggangiare nella lista */
  if ( !( tmp = ( CODA * ) malloc ( sizeof( CODA ) ) ) )
  {
```

```
61:     printf ( "\nErrore di allocazione, terminare il programma\n" );
62:     system ( "PAUSE" );
63:   }
64:   tmp->intero = intero;
65:   ( testa->prev )->next = tmp;
66:   tmp->next = testa;
67:   tmp->prev = testa->prev;
68:   testa->prev = tmp;
69:
70:   return testa;
71: }
72:
73: /* elimina un elemento dalla coda ( in testa alla coda )
74:
75:   INPUT:  size - not used
76:           *testa - puntatore al nodo sentinella
77:   OUTPUT: ritorna il valore dell'elemento eliminato
78: */
79: int Dequeue ( CODA *testa, int size )
80: {
81:   CODA* tmp; /* puntatore al nodo che sarà eliminato */
82:   int numero; /* valore che viene ritornato al chiamante */
83:
84:   tmp = testa->next;
85:   numero = tmp->intero;
86:   testa->next = ( testa->next )->next;
87:   ( tmp->next )->prev = testa;
88:
89:   free ( tmp );
90:   return numero;
91: }
92:
93: /* stampa la coda
94:
95:   INPUT:  *testa - puntatore al nodo sentinella
96:   OUTPUT:
97: */
98: void StampaCoda ( CODA *testa)
99: {
100:   CODA *tmp; /* puntatore che funziona da testina di stampa */
101:   tmp = testa->next;
102:
103:   while ( tmp != testa ) /* finchè non si raggiunge la sentinella */
104:   {
105:     printf ( "%d\t", tmp->intero );
106:     tmp = tmp->next;
107:   }
108:   printf ( "\n" );
109: }
110:
```

```c
/*                          CODA_LISTA.H
REALIZZATO DA:
Marco Sommella 50/482
Giovanni Di Cecca 50/887
Virginia Bellino 408/466

RESPONSABILI: Sommella - Bellino
*/

/* libreria per gestire una coda come
  lista puntata circolare doppia con sentinella */

typedef struct coda
{
  struct coda *prev;
  int intero;
  struct coda *next;
} CODA;

/* prototipi */
CODA *CreaCoda ( int size );

int CodaVuota ( CODA *testa );

CODA *Enqueue ( CODA *testa, int intero, int *size );

int Dequeue ( CODA *testa, int size );

void StampaCoda ( CODA *testa );

```