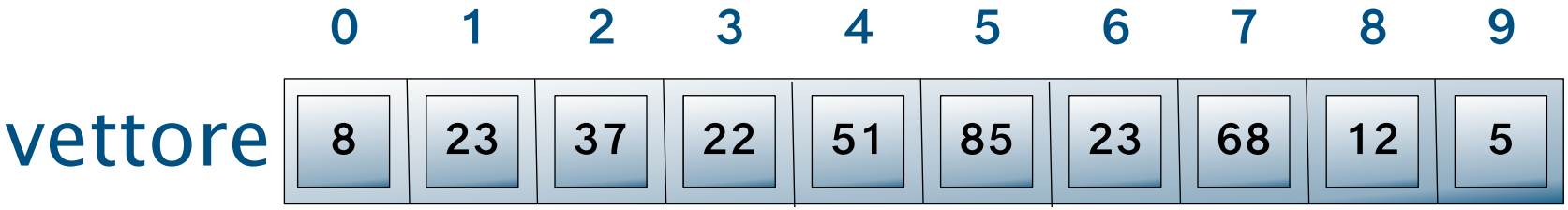
Ricerca della prima occorrenza di un elemento in un vettore tramite funzione.
Il vettore è globale.
In caso di esito positivo della ricerca, nella cella relativa all'elemento trovato bisogna inserire la differenza, in valore assoluto, tra l'elemento precedente e quello successivo ad esso, oppure 0 in caso di errore.

Esercizio F_10 - Prima occorrenza



Inserisci il valore da ricercare: 22

Elemento presente nella posizione 3

Modifica: vettore[3]= 14

Esercizio F_10 - Prima occorrenza


```c
#include <stdio.h>
#define MAX 10
#define NOT_FOUND -1
int vettore[MAX];                /* variabile array globale */
int ricerca(int ricercato); /*dichiarazione della funzione ricerca*/
int differenza(int prec, int succ);
void main() {
   int i, numero, posizione;

   for (i=0;i<MAX;i++) { /*generazione casuale dei numeri (da 0 a 99)*/
        vettore[i]=rand() % 99;
        printf("%d\t",vettore[i]);
   }
   printf("\nInserisci il valore da ricercare: ");/*acquisizione del numero*/
   scanf("%d",&numero);
   posizione =  ricerca(numero);                                  /*ricerca*/
   if (posizione==NOT_FOUND)                    /*verifica risultato ricerca*/
          printf("\nElemento non trovato!");
   else {
          printf("\nElem. presente nella pos %d", posizione);
          /*calcolo della differenza*/
          vettore[posizione] =  differenza((posizione - 1), (posizione + 1));
          printf("\nModifica: vettore[%d]=%d\n", posizione, vettore[posizione]);
   }
}
```

Esercizio F_10 - Prima occorrenza


```
/*Definizione della funzione ricerca*/
int ricerca(int ricercato)
{

   int j;
   /*Ciclo di scansione dell'array*/

   for (j=0;j<MAX;j++)
       if (vettore[j] == ricercato)
           return (j); /*Posizione dell'elemento trovato*/

   /*Scansione completa dell'array*/
   return (NOT_FOUND);     /*Codice di errore*/


}
```

Esercizio F_10 - Prima occorrenza

```
/*Definizione della funzione differenza*/
int differenza (int prec, int succ)
{
    int diff;
    /* verifica validità cursori specificati */
    if ( (prec<0) || (succ>=MAX) )
       return(0);
    /* calcola differenza */
    if (vettore[prec] > vettore[succ])
       diff = vettore[prec] - vettore[succ];
    else
       diff = vettore[succ] - vettore[prec];
    /* restituisce differenza in valore assoluto */
    return (diff);
            //diff=abs(vettore[prec] - vettore[succ]);
}
```

Scrivere un programma che verifichi la ripetizione di un elemento in un array definito globalmente sfruttando una funzione che trova la prima occorrenza dell'elemento "corrente" nelle posizioni successive dell'array se presente.

Esercizio F_11 - Cerca ripetizione

0	1	2	3	4	5	6	7	8	9
8	23	37	22	51	85	23	68	12	5
i=0									

ricerca(numero = vett[0] = 8, inizio = 1);

8	23	37	22	51	85	23	68	12	5
	j=1								
8	23	37	22	51	85	23	68	12	5

...

8	23	37	22	51	85	23	68	12	5
								j=8	
8	23	37	22	51	85	23	68	12	5
									j=9

return(-1);

Esercizio F_11 - Cerca ripetizione

0	1	2	3	4	5	6	7	8	9
8	23	37	22	51	85	23	68	12	5
	i=1								

ricerca(numero = vett[1] = 23, inizio = 2);

8	23	37	22	51	85	23	68	12	5
		j=2							
8	23	37	22	51	85	23	68	12	5
			j=3						

...

8	23	37	22	51	85	23	68	12	5
						j=6			

return(j = 6);

Esercizio F_11 - Cerca ripetizione

```
#include <stdio.h>
#define MAX 10
int vett[MAX];                    /* variabile array globale */
/* Ricerca di un elemento in un vettore globale; restituisce
la posizione dove viene trovato l'elemento ricercato o -1 se
non lo trova.
Parametri:
 numero: numero da ricercare nell'array
 inizio: posizione iniziale dalla quale iniziare la ricerca*/

 int ricerca(int numero, int inizio) {

          int j;
          for( j = inizio ; j<MAX; j++ )
             if ( numero == vett[j] )
                  return(j);
          return(-1);

 }
```

Esercizio F_11 - Cerca ripetizione

```
void main() {
  int i, k;
  printf("\nInserisci il vettore di Numeri :");
  for ( i=0; i<MAX; i++ ) {
       printf("\nInserisci il %d elemento: ",i);
       scanf("%d",&vett[i]);
  }
  for (i=0; i<(MAX-1); i++) { /* verifica ripetizioni*/
       k = ricerca(vett[i], i+1);
      if ( k > 0 )
         printf("Gli elementi %d e %d coincidono",i,k);
  }
}
```

Esercizio F_12 - Pari o Dispari

Scrivere un programma che, generato un numero casuale, determini se esso è pari oppure dispari e lo memorizzi tramite un'apposita funzione in uno di due array (array dei pari ed array dei dispari).

Esercizio F_12 - Pari o Dispari


```c
#include <stdio.h>
#define MAX 10          /* dimensione dei vettori */
/* inserisce il parametro numero nell'array vett in una cella
                                        contenente -1, se esiste*/
int ins_num(int vett[], int numero) {

    int i;
    for ( i=0; i<MAX; i++ ) /* ricerca cella disponibile */
        if ( vett[i] == -1 ) {
            vett[i]=numero;
            /* esito dell'inserimento in caso positivo */
            return (0);
        }
    return(-1);     /* esito dell'inserimento in caso negativo */

}

void init(int vett[]) {              /* inizializza un vettore */

    int i;
    for ( i=0; i<MAX; i++ )      /* ricerca cella disponibile */
         vett[i]=-1;
}
```

Esercizio F_12 - Pari o Dispari

```
void main() {
    int num, stop=0;
    int vett_pari[MAX], vett_dispari[MAX];
    /* inizializzazione */
    init(vett_pari);
    init(vett_dispari);
    srand(time(NULL));
    do{
        num=rand() % 100; /* numero casuale */
        if ( num%2 == 0 ) /* inserimento nell'apposito array*/
            stop=ins_num(vett_pari, num);
        else
            stop=ins_num(vett_dispari, num);
    } while ( stop == 0 );
    for ( num=0; num<MAX; num++ ) /* visualizzazione */
        printf("\n pos %2d: pari %2d, dispari %2d",num,
                        vett_pari[num], vett_dispari[num]);
}
```

Ordinamento per selezione tramite funzioni.

Esercizio F_13 - Ordinamento per selezione

```
/* Prototipi delle funzioni: */

/* ordina il vettore ricevuto come parametro */
void ordina(int vett[]);


/* scambia due interi; in questo esercizio i due interi
                       sono due elementi di un vettore */
void scambio(int *a, int *b)


/* trova l'elemento minore in un vettore partendo
                          da una posizione specificata */
int minore( int vett[ ] ,int iniz )
```

Esercizio F_13 - Ordinamento per selezione


```c
#define MAX 20
void main() {
    int i, vett[MAX];
    /* popolamento del vettore con numeri casuali */
    srand(time(NULL));
    printf("\nVettore Originale:");
    for ( i=0; i<MAX; i++ ) {
        vett[i]=rand() % 100;
        printf("\nVett[%d] = %d",i,vett[i]);
    }
    ordina(vett); /* ordinamento del vettore */
    printf("\n\nVettore Ordinato:");
    /* stampa del vettore finale ordinato */
    for( i=0; i<MAX; i++ )
        printf("\nVett[%d] = %d",i,vett[i]);
}
```

Esercizio F_13 - Ordinamento per selezione

```
/* ordina il vettore ricevuto come parametro */
void ordina(int vett[]) {

    int i, minpos;
    /* ordinamento per selezione */
    for ( i=0; i<(MAX-1); i++ )
    {
            /* trova l'elemento minore */
            minpos=minore(vett,i);

            /* eventualmente scambia gli elementi */
            if ( minpos != i )
               scambio(&vett[i],&vett[minpos]);
    }

}
```

Esercizio F_13 - Ordinamento per selezione


```
/* scambia due interi; in questo esercizio i
due interi sono due elementi di un vettore */

void scambio(int *a, int *b) {
      int appoggio;

      /* scambio */
      appoggio=*a;
      *a=*b;
      *b=appoggio;



}
```

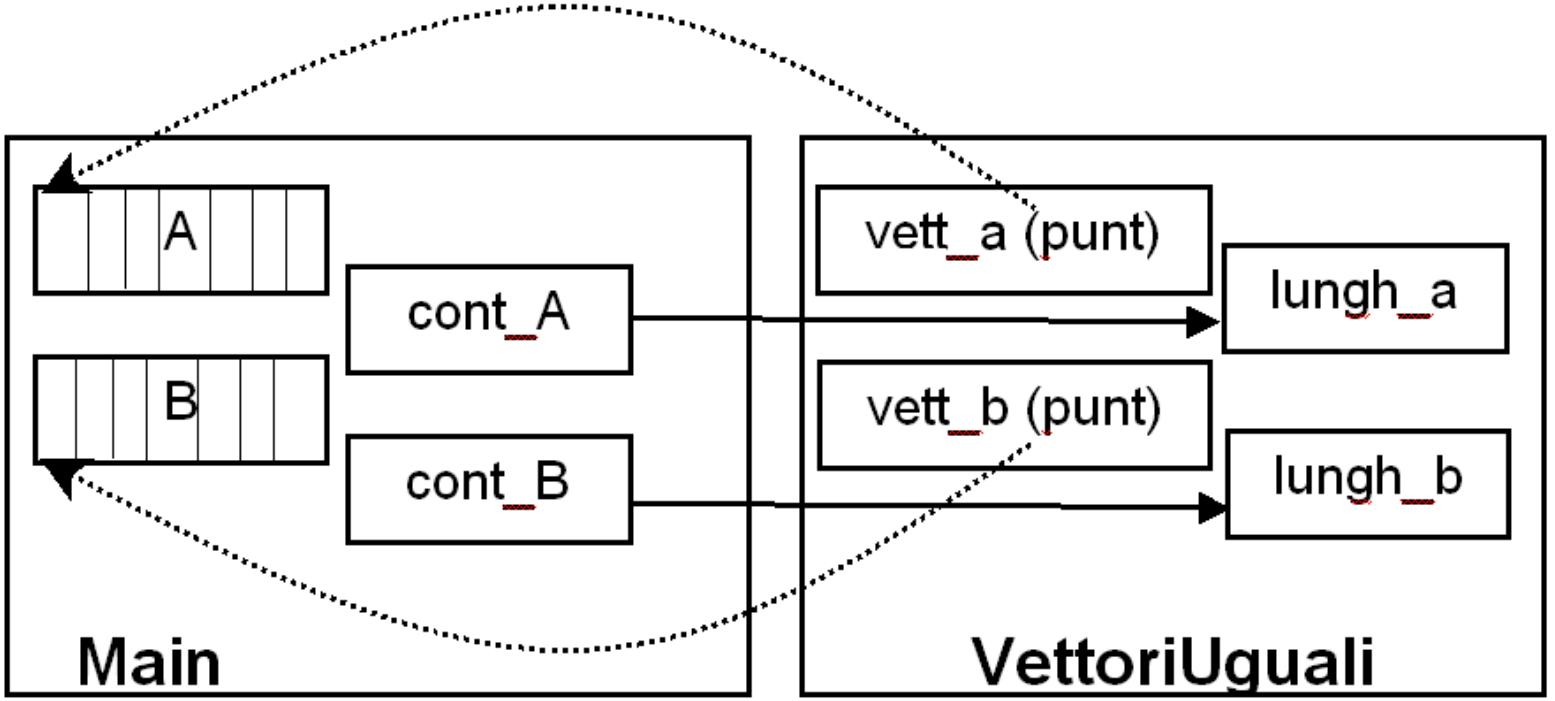
Esercizio F_13 - Ordinamento per selezione

```
/* trova l'elemento minore in un vettore partendo
    da una posizione specificata */
int minore( int vett[ ] ,int iniz ) {
  int j,
  int minpos;     /* posizione dell'intero minore */
  /* inizializzazione del risultato (al primo elemento
                                          utile) */
  minpos=iniz;

  /* ricerca */
  for (j=iniz+1;j<MAX;j++)
        if ( vett[ j ] < vett[minpos] )
           minpos = j;

  /* restituisce il risultato */
  return(minpos);
}
```

Esercizio F_14 - Confronto di vettori

Scrivere una funzione che riceva 2 array di interi e la loro lunghezza e restituisca "1" se essi hanno lo stesso contenuto.

Esercizio F_14 - Confronto di vettori



Esercizio F_14 - Confronto di vettori


```c
#define MAX_DIM 50

/* funzione che riceve e confronta i due vettori restituisce 1 nel caso il
contenuto dei due vettori sia identico */


int VettoriUguali(int vett_a[ ], int lungh_a, int vett_b[ ], int lungh_b)
{

        int i;
        if ( lungh_a != lungh_b )
                return (0);

        for ( i=0; i<lungh_a; i++ )
                if ( vett_a[i] != vett_b[i] )
                        return (0);

        return (1);
}
```

Esercizio F_14 - Confronto di vettori


```
void main()
{

  int A[MAX_DIM], B[MAX_DIM];
  int cont_A=0, cont_B=0;

  printf("\nInserisci il primo numero per A:");
  scanf("%d", &A[cont_A]);

  while (A[cont_A] != -1)
  {
        cont_A++;
        printf("\nInserisci il prox numero per A:");
        scanf("%d", &A[cont_A]);
  }

                    /*...*/
```

Esercizio F_14 - Confronto di vettori


```
/*...*/

  printf("\nInserisci il primo numero per B:");
  scanf("%d", &B[cont_B]);
  while ( B[cont_B] != -1 ) {
        cont_B++;
        printf("\nInserisci il prox numero per B:");
        scanf("%d", &B[cont_B]);
  }

  if (VettoriUguali(A, cont_A, B, cont_B) == 1 )
        printf("\nI vettori contengono gli stessi dati!");
  else
        printf("\nI vettori contengono dati diversi!");
}
```

Scrivere una funzione che calcoli la lunghezza di una stringa passata come parametro, sfruttando l'aritmetica dei puntatori.

Esercizio F_15 - Lunghezza stringa


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

int lungh(char *);                          /* prototipo */

void main()
{
  char stringa[100];

  printf("\nInserire la str. da analizzare (senza spazi): ");
  fflush(stdin);
  scanf("%s", stringa);

  printf("\nLa lungh. della str. e' : %d",  lungh(stringa) );
  printf("\nLa stringa originale e' %s", stringa);


}
```

Esercizio F_15 - Lunghezza stringa


```
/* funzione per calcolare la lunghezza della stringa */
int lungh(char *stri)
{
  int cont=0;

  /* scans. della stringa fino al carattere
                                        terminatore '\0' */
  while (*stri != '\0')
  {
       stri++;
       cont++;
  }

  /* restituzione del valore */
  return (cont);
}
```

Scrivere una funzione che calcoli la percentuale di lettere e di cifre numeriche presenti nella stringa passata come parametro.
Il risultato (ovvero le due percentuali) viene restituito utilizzando un'unica struttura.

Esercizio F_16 - Percentuali di stringa


```
#define MAX 80
typedef struct {
        float carat;
        float cifre;
} analisi; /* tipo per contenere il risultato dell'analisi della stringa */
void conteggio(char [ ], analisi *);     /* prototipo */

void main() {
   analisi risultato;
   char stringa[MAX];
   printf("\nInserisci la stringa da analizzare: ");
   scanf("%s",stringa);          /* inserimento stringa */
   risultato.cifre=0.0;
   risultato.carat=0.0;             /* inizializzazione */
   conteggio(stringa, &risultato); /* analisi della stringa */
   /* visualizzazione risultato */
   printf("\nPercentuale di cifre = %5.2f%% e di caratteri =
                  %5.2f%%", risultato.cifre, risultato.carat);
}
```

Esercizio F_16 - Percentuali di stringa


```
/* la funzione conta le lettere e le cifre che compongono  *
 * la stringa passata come parametro e calcola le relative *
 * percentuali */
void conteggio(char sequenza[], analisi *risultato) {
  int i;

  … /* codice controllo validità della sequenza di caratteri */

  for (i=0; i<strlen(sequenza); i++)

     /* verifica di ciascun carattere */
     if ( (sequenza[i] >= '0') && (sequenza[i] <= '9') )
         risultato->cifre = risultato->cifre+1;
     else
         risultato->carat = risultato->carat+1;

  /* calcolo delle percentuali */
  risultato->cifre = (risultato->cifre / strlen(sequenza)) * 100;
  risultato->carat = (risultato->carat / strlen(sequenza)) * 100;
}
```

Esercizio F_17 - To Upper

Implementare una funzione che converta in maiuscolo una stringa passata come parametro.

Esercizio F_17 - To Upper


```
#define NUM 3
#define MAX_STR 50
struct {
        char cognome[MAX_STR];
        char nome[MAX_STR];
        char indirizzo[MAX_STR];
        char citta[MAX_STR];
        int  anno_nascita;
} anagrafe[NUM];

/* converte in maiusc. la stringa passata come parametro*/
void maiuscole(char str[ ]) {
   int x;
   for (x=0; x < strlen(str); x++ )
           if ( (str[x] >= 'a') && (str[x] <= 'z') )
              str[x] = str[x] + ('A' - 'a');
}
```

Esercizio F_17 - To Upper

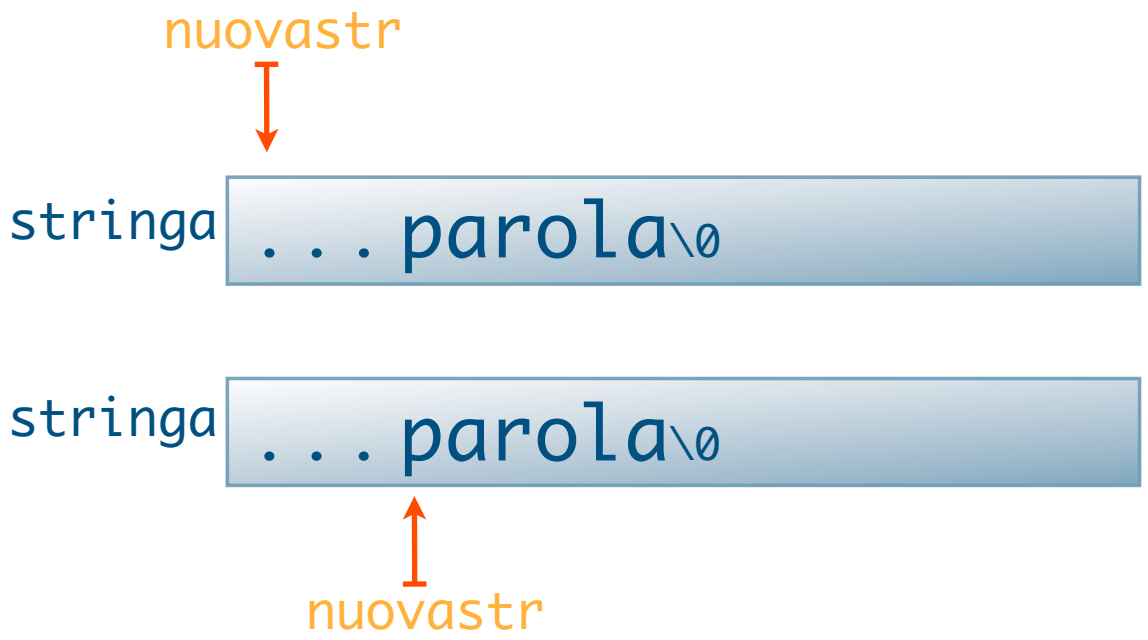

```
int i;
void main() {
        /* inserimento anagrafe */
        ...

        /* conversione in stringhe maiuscole */
        for (i=0; i<NUM; i++) {
                maiuscole(anagrafe[i].cognome);
                maiuscole(anagrafe[i].nome);
                maiuscole(anagrafe[i].indirizzo);
                maiuscole(anagrafe[i].citta);
        }
}
```

Esercizio F_18 - Elimina punti

Scrivere una funzione che, riceva come parametro una stringa e, nel caso essa inizi con dei punti, la modifichi eliminando i punti stessi, e restituisca il numero di punti tolti.

Esercizio F_18 - Elimina punti


```
/* Chiamata: */

char stringa[MAX];                  /* vettore allocato staticamente */
char *nuovastr;                                 /* solo un puntatore */
nuovastr = stringa;            /* puntatore all'inizio della stringa */
punti = toglipunti(&nuovastr);     /* eliminaz. dei punti iniziali */

printf("\nHo tolto %d punti", punti );
printf("\nLa stringa modificata è %s", nuovastr );
```



Esercizio F_18 - Elimina punti


```
#define MAX 80
/*dalla stringa passata come parametro vengono eliminati gli
eventuali punti. Il numero di questi punti viene restituito*/

int toglipunti(char **stringa) {
   int punti=0;

   /* elimina i punti iniziali */
   while ( *stringa[0] == '.' && punti < MAX ) {
      /* incrementa contatore */
      punti=punti+1;

      /*incrementa puntatore originale (passato per indirizzo)*/
      *stringa=*stringa+1;
   }

   /* restituisce i punti eliminati */
   return(punti);
}
```

Definire le strutture dati e le funzioni necessarie (inserimento e cancellazione libri) per gestire molteplici biblioteche. Una biblioteca contiene uno scaffale composto da vari libri (possono esserci posizioni vuote).
Ogni libro è caratterizzato da autore e titolo.

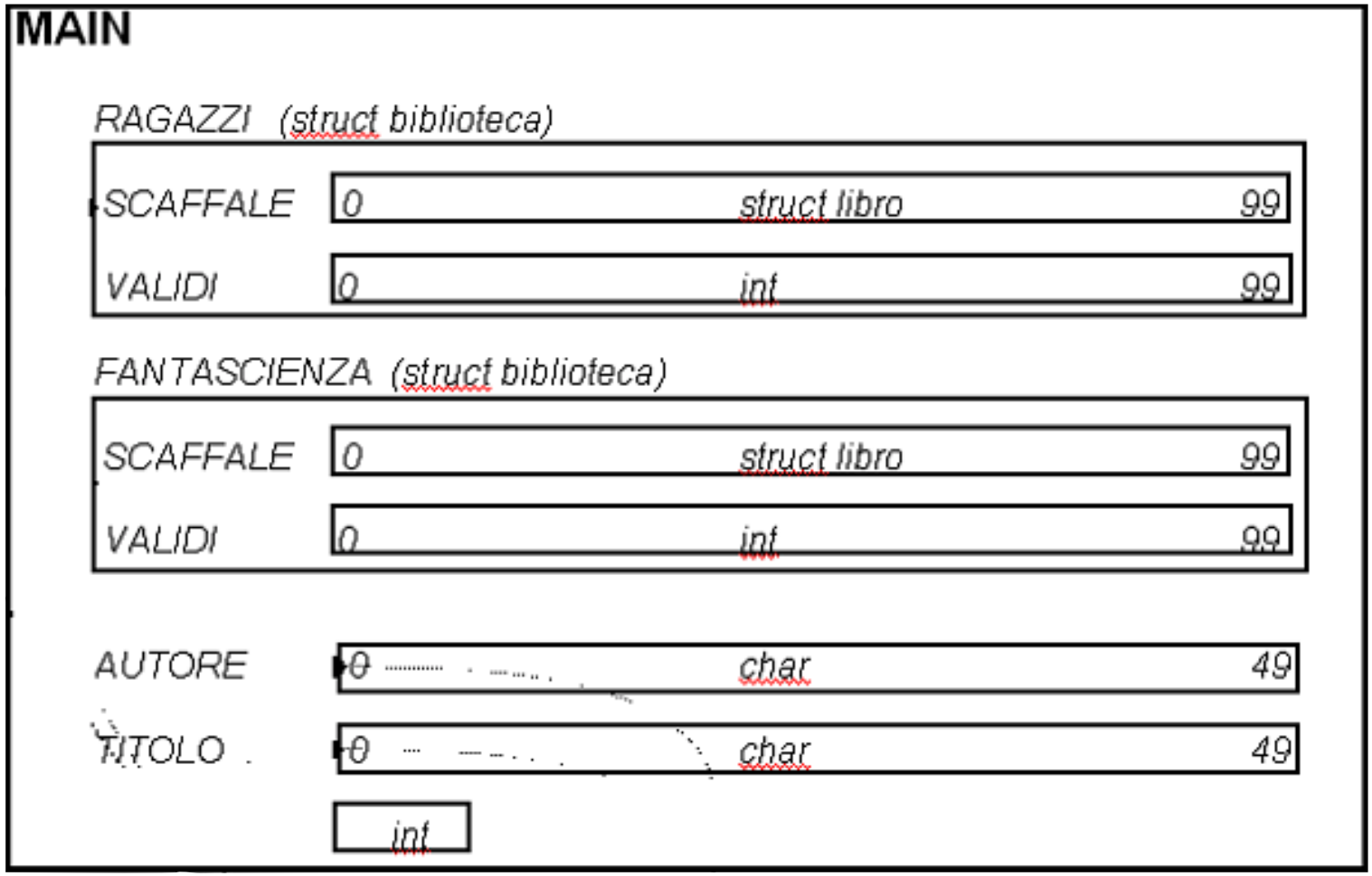
Esercizio F_19 - Biblioteca

```c
#define MAX_LIBRI 100
#define MAX_STR 50

typedef enum {false, true} boolean;


typedef struct {
  char autore[MAX_STR];
  char titolo[MAX_STR];
} Libro;


typedef struct {
  Libro scaffale[MAX_LIBRI];
  boolean validi[MAX_LIBRI];  /* per ogni libro indica
                                 se è valido */
} Biblioteca;
```

Esercizio F_19 - Biblioteca

```
void main()
{
  Biblioteca Ragazzi, Fantascienza;
  char autore[MAX_STR], titolo[MAX_STR];
  int i;

  for (i=0; i<MAX_LIBRI; i++)
  {
     Ragazzi.validi[i] = false;
     Fantascienza.validi[i] = false;
  }

        /* Il main() prosegue ... */
```

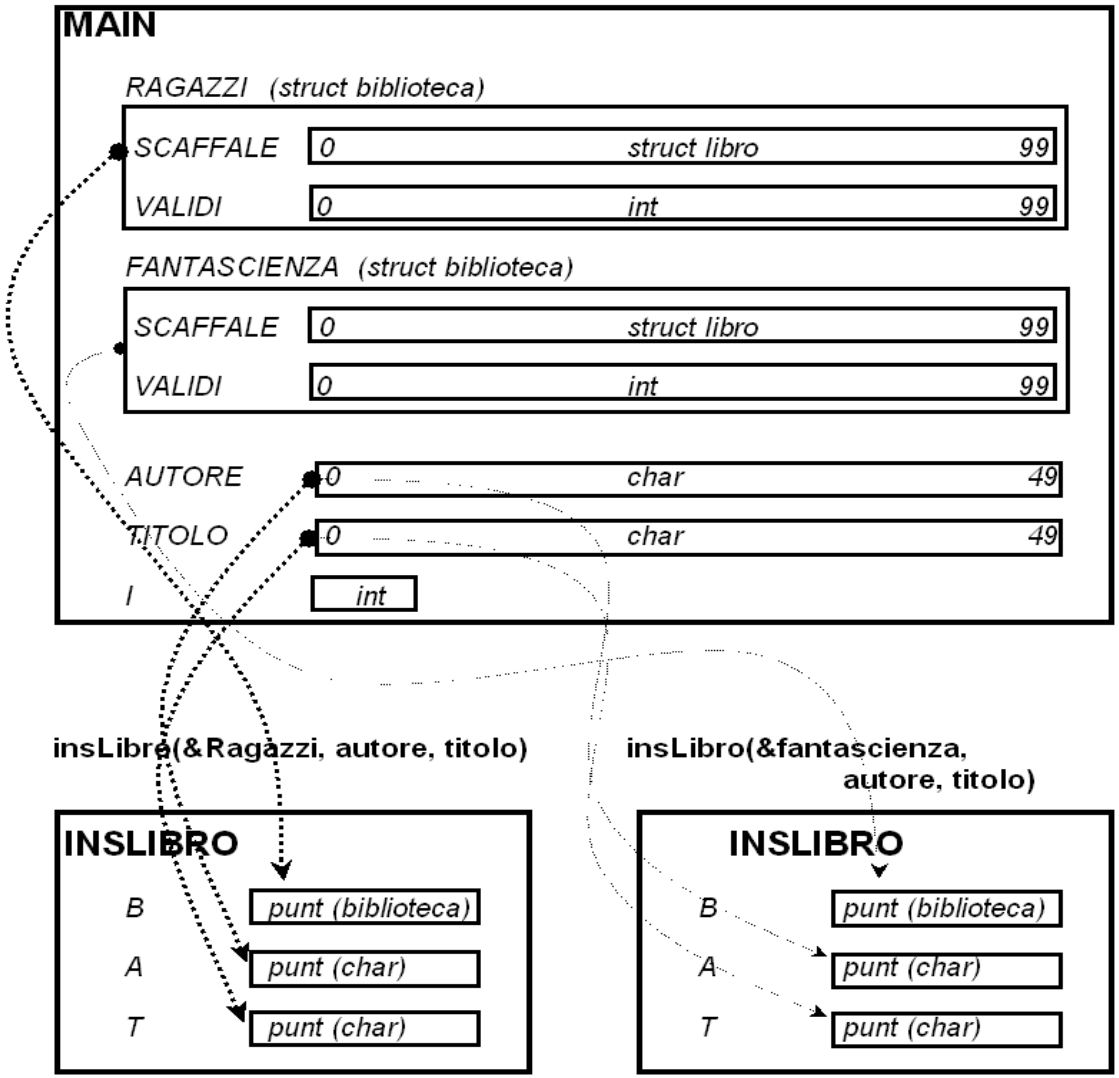
Esercizio F_19 - Biblioteca



Esercizio F_19 - Biblioteca

```
                /* …continua il main() */
   strcpy(autore,"Rowling");
   strcpy(titolo, "Harry Potter");
   if (   insLibro(&Ragazzi, autore, titolo)     == 0 )
   {
       printf("Errore\n");
       exit(1);
   }
   strcpy(autore,"Asimov");
   strcpy(titolo, "Fondazione e Terra");
   if ( insLibro(&Fantascienza, autore, titolo) == 0 )
   {
       printf("Errore\n");
       exit(1);
   }
}
```

Esercizio F_19 - Biblioteca



testata della funzione :

int insLibro(Biblioteca * b, char a[], char t[])

Esercizio F_19 - Biblioteca


```
/* la Biblioteca va passata per indirizzo perché verrà modificata. Ritorna
zero se c'e` qualche errore, altrimenti il numero di libri nella biblioteca*/
int insLibro(Biblioteca * b,  char a[ ], char t[ ]) {
        int i = 0;
        int cont = 0;
        boolean inserito = false;

        for (i=0; i<MAX_LIBRI; i++) {
                if ( inserito==false && b->validi[i] == false ) {
                        /* ho trovato un posto libero */
                        strcpy(b->scaffale[i].autore, a);
                        strcpy(b->scaffale[i].titolo, t);
                        b->validi[i] = true;
                        inserito = true;
                }
               if ( b->validi[i]==true )
                        cont++;
        }
        if (inserito == false)
                return 0;
        else
                return cont;
}
```

Esercizio F_19 - Biblioteca

```
boolean delLibro(Biblioteca * b,  char a[ ], char t[ ])
{

   int i=0;
   boolean trovato = false;
   while ( trovato==false && i<MAX_LIBRI ) {
      if ( b->validi[i] == true
           && strcmp(b->scaffale[i].autore, a) == 0
           && strcmp(b->scaffale[i].titolo, t ) == 0 )
      {
                  b->validi[i] = false;
                  trovato = true;
      }

      i++;
   }
   return trovato;

}
```

Implementare le funzioni necessarie ad inserire ed eliminare interi da un archivio con stato pieno/vuoto.
Il vettore dell'archivio viene passato come parametro.

Esercizio F_20 - Archivio

Si consideri il problema di memorizzare N numeri interi in un archivio, sfruttando un apposito flag per memorizzare lo stato delle singole celle (usata, libera).
Si definiscano le strutture dati richieste per generare tale archivio
Si implementino, tramite apposite funzioni/procedure, le seguenti funzionalità:

1. Aggiunta di un elemento all'archivio
2. Eliminazione di un elemento dall'archivio
3. Visualizzazione degli elementi presenti nell'archivio
4. Ricerca di un elemento nell'archivio
5. Inserimento di un elemento solo se esso non è già presente nell'archivio
6. Incremento di una unità degli elementi presenti nell'archivio
7. Raggruppamento degli elementi validi all'inizio dell'archivio

Ottimizzare il codice implementando funzioni che possono essere utilizzate in più punti del programma.

Esercizio F_20 - Archivio

```
#define MAX 10

typedef enum {VUOTA, PIENA} dispon;
typedef struct {
        int elemento;
        dispon stato;
} cella;

/* inizializzazione */
void init(cella *lista) {
        int i;

        for ( i=0; i<MAX; i++ )
                lista[i].stato=VUOTA;
}
```

Esercizio F_20 - Archivio


```
/* inserimento di un numero in una cella disponibile */
int  ins (   cella *seq, int numero    ) {

      int i;
      for(i=0;i<MAX;i++) {    /* ricerca cella disponibile */
                  if ( seq[i].stato == VUOTA ) {
                        seq[i].elemento=numero;
                        seq[i].stato=PIENA;
                        return (i);
                  }
      }
      return (-1);     /* errore: array pieno */


}

/* inserimento di un numero in una cella (solo se l'elemento non è già presente) */
int ins_esclusivo (   cella *seq, int numero   ) {

      int i;
      if ( esiste(seq, numero) == -1 )
            return ins(seq, numero);
      return (-3);     /* errore: elemento già presente */
}
```

Esercizio F_20 - Archivio


```
int esiste (    cella *elem, int numero          ) {
  int i;
  for( i=0; i<MAX; i++ ) { /* ricerca cella disponibile */
  if ( (elem[i].stato==PIENA) && (elem[i].elemento==numero) )
          return (i);
  }
  /* elemento non trovato */
  return (-1);

}

int del(      cella *array,int numero                  ) {
  int pos;
  if ( (pos=esiste(array, numero)) != -1) {/* ricerca dell'elemento*/
          array[pos].stato=VUOTA; /* eliminazione */
          return (pos);
  }
  /* errore: elemento non trovato */
  return (-2);
}
```

Esercizio F_20 - Archivio

```
/* incrementa di una unità gli elementi validi */
int incrementa (     cella *elem          ) {

   int i, validi=0;
   for( i=0; i<MAX; i++ )
   {
      if ( elem[i].stato == PIENA )
      {
         elem[i].elemento++;
         validi++;
      }
   }

   return (validi);
   /* restituisce conteggio degli elementi validi */

}
```

Esercizio F_20 - Archivio


```
/* restituisce il primo elemento libero */
int primolibero (   cella *vett                    ) {

   int i;
   for( i=0; i<MAX; i++ ) /* scansione dell'array */
           if ( vett[i].stato == VUOTA )
              return (i); /* restituisce posizione i */

   return (-1);      /* nessun elemento libero */
}

/* restituisce l'ultimo elemento valido */
int ultimovalido (   cella *vett                    ) {

   int i;
   for( i=MAX-1; i>=0; i-- ) /* scansione dell'array */
           if ( vett[i].stato == PIENA )
              return (i); /* restituisce posizione i */

   return (-1);     /* nessun elemento libero */
}
```

Esercizio F_20 - Archivio


```
/* compatta le celle contenenti elementi validi */
void compatta (  cella *vett                    ) {

    int libero, valido;
    /* recupera le celle coinvolte */
    libero = primolibero(vett);
    valido = ultimovalido(vett);

    /* eventuale scambio tra un elemento vuoto e uno libero */
    while ( (libero<valido) && !( (libero==-1) || (valido==-1) ))
    {
       /* scambio */
       vett[libero]=vett[valido];
       vett[valido].stato=VUOTA;
       /* recupera le celle coinvolte */
       libero = primolibero(vett);
       valido = ultimovalido(vett);
    }

}
```

Esercizio F_20 - Archivio

```
/* ulteriore ottimizzazione: riutilizzo della funzione
   primolibero() */

/*inserimento di un numero in una cella disponibile*/
int ins (  cella *seq, int numero                ) {

      int i;
      i = primolibero(seq);

      if (i >= 0)
      {              /* inserimento */
              seq[i].elemento=numero;
              seq[i].stato=PIENA;
              return (i);
      }
      else
              return (-1);     /* errore: array pieno */

}
```

Esercizio F_20 - Archivio


```
void main()
{
  cella vett[MAX];
  int operaz=-1, numero, posiz, ris;

  /* inizializzazione */
  init(vett);
  while ( operaz!=0 )
  { /* richiesta operazione */
        /* richiesta operazione da eseguire */
        printf("\nInserisci l'operazione:");
        printf("\n\t0 fine");
        printf("\n\t1 aggiungi");
        printf("\n\t2 elimina");
        printf("\n\t3 visualizza");
        printf("\n\t4 ricerca");
        printf("\n\t5 inserisci esclusivo");
        printf("\n\t6 incrementa");
```

Esercizio F_20 - Archivio

```
printf("\n\t7 compatta");
printf("\n\t: ");
scanf("%d",&operaz);

/* elaborazione */
switch ( operaz )
{
  case 0:
        printf("\nFine Programma!");
   break;
  case 1: /* inserimento */
        /* richiesta numero sul quale operare */
        printf("\nInserisci il numero positivo: ");
        scanf("%d",&numero);
        if (ins(vett,numero)==-1)
           printf("Non ci sono celle disponibili!")
   break;
```

Esercizio F_20 - Archivio

```
case 2: /* richiesta numero sul quale operare */
        printf("\nInserisci il numero positivo: ");
        scanf("%d",&numero);
        if (del(vett,numero)==-2) /* eliminazione */
          printf("Il numero richiesto non e' stato
                                          trovato!");
  break;

case 3: /* visualizzazione array */
        vis(vett);
  break;
```

Esercizio F_20 - Archivio

```
case 4: /* ricerca elemento */

        /* richiesta numero da ricercare */
        printf("\nInserisci il numero positivo: ");
        scanf("%d",&numero);

        /* ricerca */
        if ( (posiz=esiste(vett,numero)) == -1 )
           printf("Il numero richiesto non è stato
                                              trovato!");
        else
           printf("Il numero richiesto è nella
                                    posizione %d!", posiz);
  break;
```

Esercizio F_20 - Archivio

```
case 5: /* inserimento esclusivo */

        /* richiesta numero da inserire */
        printf("\nInserisci il numero positivo: ");
        scanf("%d",&numero);
        ris = ins_esclusivo(vett,numero);
        switch (ris) {
                case -1:
                        printf("Non ci sono celle
                                        disponibili!");
                  break;

                case -3:
                        printf("Elemento già presente!");
                  break;
        }
   break;
```

Esercizio F_20 - Archivio

```
    case 6: /* visualizzazione array */
            printf("ho incrementato %d elementi!",
                                        incrementa(vett));
      break;

    case 7: /* visualizzazione array */
            compatta(vett);
      break;

    default:
            printf("\nOperazione non supportata!");
      break;

    }
  }
}
```