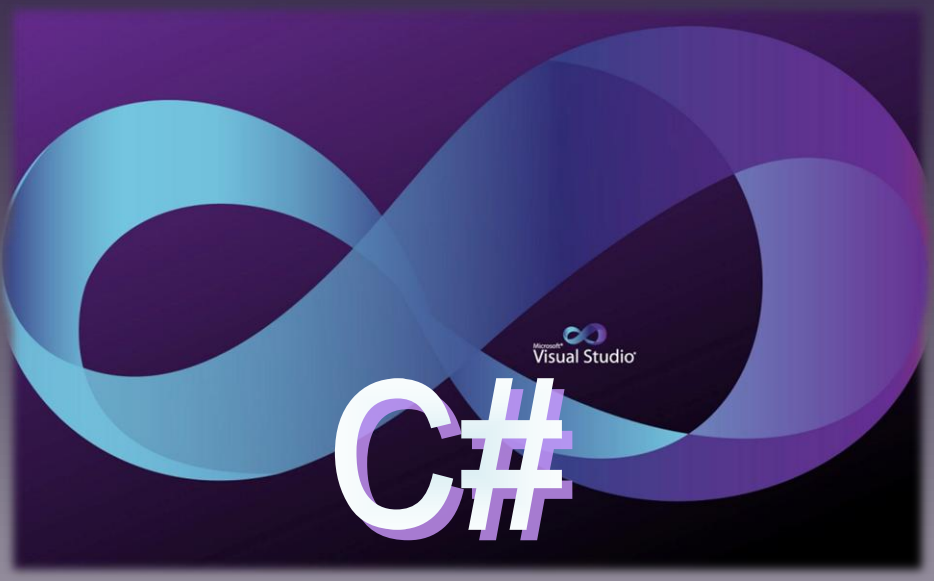
ISTITUTO TECNICO INDUSTRIALE
G. M. ANGIOY
SASSARI



CORSO DI PROGRAMMAZIONE

ALGORITMI SUI VETTORI

DISPENSA 05.02

05-02_Algoritmi_Vettori_[ver_15]

Questa dispensa è rilasciata sotto la licenza Creative Common CC BY-NC-SA. Chiunque può copiare, distribuire, modificare, creare opere derivate dall'originale, ma non a scopi commerciali, a condizione che venga riconosciuta la paternità dell'opera all'autore e che alla nuova opera vengano attribuite le stesse licenze dell'originale.

Versione del: 07/11/2015
Revisione numero: 15

Prof. Andrea Zoccheddu
Dipartimento di Informatica

DIPARTIMENTO
INFORMATICA E TELECOMUNICAZIONI

ALGORITMI SUI VETTORI

ALGORITMI SUI VETTORI

INTRODUZIONE AI VETTORI

PROGETTO GUIDATO

- Si prepari un Form1 simile alla figura
- Preparare le variabili globali seguenti

```
int[] pippo = new int[100];
int[] poldo = new int[100];
```

- Impostare le seguenti proprietà della etichetta **label1**:
  - AutoSize → False;
  - TextAlign → MiddleCenter;
- Vogliamo predisporre il pulsante **Genera** per allocare memoria dei vettori pippo e poldo e inizializzarne i valori delle celle; fai doppio clic su Genera e associa il seguente codice:

```
Random R = new Random();
listBox1.Items.Clear();      //pulisce la listBox1 dal suo contenuto
listBox2.Items.Clear();      //pulisce la listBox2 dal suo contenuto
for (int i = 0; i < pippo.Length; i++)
{
     pippo[i] = R.Next(10000);     //inizializzazione cella di pippo
     poldo[i] = R.Next(10000);     //inizializzazione cella di poldo
     listBox1.Items.Add(pippo[i]);
     listBox2.Items.Add(poldo[i]);
}
```

- Il pulsante **Minimo** dovrà cercare il più piccolo valore di pippo e mostrarlo in butto2; fai doppio clic su Minimo e associa il seguente codice:

```
int min = pippo[0];
for (int i = 0; i < pippo.Length; i++)
{
     if (min > pippo[i])
     min = pippo[i];
     button2.Text = "minimo= " + Convert.ToString(min);
}
```

- Il pulsante **Cerca** dovrà verificare se un determinato valore è presente dentro il vettore pippo e mostrarlo in label1; fai doppio clic su Cerca e associa il seguente codice:

```
int x = Convert.ToInt16(textBox1.Text);        //valore da cercare
bool trovato = false;
int i = 0;
while ((i < pippo.Length)&&(!trovato))
{
     if (x == pippo[i])
     trovato = true;                           //l'ho trovato in posizione i
     else i++; //altrimenti vado avanti
}
if (trovato)
     label1.Text = "TROVATO";
else
     label1.Text = "NON C'E'";
if (trovato)
     listBox1.SelectedIndex = i;
else
     listBox1.SelectedIndex = -1;
```

- Il pulsante **Ordina1** dovrà ordinare il vettore pippo e visualizzarlo in listBox1; fai doppio clic su Ordina1 e associa il seguente codice:

```
//ordinamento ingenuo
for (int k = 0; k < pippo.Length -1; k++)
     for (int i = 0; i < pippo.Length -1; i++)
           if (pippo[i] > pippo[i + 1])
           {
                 int tmp = pippo[i];
                 pippo[i] = pippo[i + 1];
                 pippo[i + 1] = tmp;
           }
//visualizzazione
listBox1.Items.Clear();
for (int i = 0; i < pippo.Length; i++)
     listBox1.Items.Add(pippo[i]);
```

- Il pulsante **BubbleSort** dovrà ordinare il vettore pippo con un algoritmo diverso da Ordina1 e visualizzarlo in listBox1; fai doppio clic su BubbleSort e scrivi il codice:

```
//ordinamento a bolle
int ultimo = pippo.Length-1;//tutto da controllare...
int fine;
while (ultimo > 0) //mentre ci sono scambi da fare...
{
     fine = ultimo -1; //fine ricorda tutti i controlli da fare...
     ultimo = 0; //azzero gli scambi da ricordare
     for (int i = 0; i<=fine;i++)
          if (pippo[i]>pippo[i+1])//se sono da scambiare
          {   //eseguo lo scambio
               int tmp = pippo[i];
               pippo[i] = pippo[i + 1];
               pippo[i + 1] = tmp;
               ultimo = i; //mi segno lo scambio appena fatto
          }
}
for (int k = 0; k < pippo.Length -1; k++)
     for (int i = 0; i < pippo.Length -1; i++)
          if (pippo[i] > pippo[i + 1])
          {
               int tmp = pippo[i];
               pippo[i] = pippo[i + 1];
               pippo[i + 1] = tmp;
          }
//visualizzazione
listBox1.Items.Clear();
for (int i = 0; i < pippo.Length; i++)
     listBox1.Items.Add(pippo[i]);
```

- Il pulsante Varie invece è proposto per provare diversi algoritmi; devi scriverli uno alla volta, cancellando il precedente e scrivendo il successivo. Prova i seguenti algoritmi:

CALCOLO DELLA MEDIA

```
double media = 0;
for (int i = 0; i < pippo.Length; i++)
     media += pippo[i] ;
media = media / pippo.Length;
label1.Text = Convert.ToString(media);
```

COPIARE UN VETTORE NELL'ALTRO

```
for (int i = 0; i < pippo.Length; i++)
     poldo[i] = pippo[i] ;
listBox2.Items.Clear();
for (int i = 0; i < pippo.Length; i++)
     listBox2.Items.Add(poldo[i]);
```

INCREMENTARE I VALORI DEL VETTORE

```
for (int i = 0; i < pippo.Length; i++)
     poldo[i]++ ;
listBox2.Items.Clear();
for (int i = 0; i < pippo.Length; i++)
     listBox2.Items.Add(poldo[i]);
```

CONTARE QUANTI NUMERI PARI CI SONO

```
int cont = 0;
for (int i = 0; i < pippo.Length; i++)
     if ((poldo[i]%2)==0)
          cont++;
label1.Text = Convert.ToString(cont);
```

ALGORITMI FONDAMENTALI SUGLI ARRAY

NOTA: USO DELL'ISTRUZIONE BREAK PER TERMINARE I CICLI

Tutti i cicli possono essere interrotti nella loro esecuzione per forzare il trasferimento dell'esecuzione all'istruzione seguente il ciclo. L'istruzione che forza questa uscita è la **`break`**.

ALGORITMI DI RICERCA E DI ORDINAMENTO

I vettori sono importantissimi tipi di dato che vengono usati in numerosissime applicazioni, specie matematiche. Sui vettori si sono, perciò, costruiti moltissimi algoritmi ed impieghi, che vengono impiegati per gestioni di dati numerici, alfanumerici e logici. Tra gli algoritmi fondamentali dobbiamo ricordare le ricerche e gli ordinamenti.

Le ricerche sono algoritmi che esplorano il vettore o parte di esso, per verificare la presenza di valori o il rispetto di regole tra essi.

Sono esempi di ricerche:

- La ricerca del valore massimo e del valore minimo contenuto in esso (qual è il più piccolo valore?);
- La ricerca di un determinato valore contenuto in esso (è presente il numero X?);
- Il calcolo della somma e della media dei valori contenuti in un vettore
- La verifica se un vettore è già ordinato o rispetta certe condizioni (sono tutti pari i suoi valori?)

L'ordinamento invece consiste nel modificare il vettore, spostando i valori contenuti in esso in modo che rispettino un ordinamento (crescente o decrescente) ovvero che ogni coppia di valori vicini (attigui) sia ordinata.

Esistono molti algoritmi di ordinamento sui vettori tra cui vogliamo ricordare:

- L'ordinamento dell'ingenuo, che esplora il vettore più volte del necessario
- L'ordinamento a bolle (bubble sort) che è abbastanza efficiente
- L'ordinamento a fusione (merge sort) che agisce su tronconi di vettore sempre più piccoli e poi li fonde
- I primi due sono considerati fondamentali in questo corso e saranno oggetto di verifica.

MODALITÀ DI RICERCA

A questo punto è interessante riflettere sui concetti di efficacia e efficienza di un algoritmo.

Come si è potuto vedere nel corso, gli algoritmi impiegano un certo numero di passi e di risorse per giungere alla soluzione richiesta. Alcuni algoritmi impiegano più passi di altri, e alcuni impiegano più passi del necessario. Tanti meno passi (o risorse) impiega l'algoritmo, tanto più è considerato efficiente. Per chiarire meglio questo concetto, facciamo un esempio.

Supponiamo che il preside incarichi una persona di verificare se in aula magna è presente uno studente che si chiama «Abele». Per svolgere il suo compito l'incaricato entra nella sala dove ci sono circa 200 persone e qui inizia a chiedere ai vari presenti se il rispettivo nome sia Abele; dopo 15 tentativi, la persona intervistata risponde «si, mi chiamo Abele». A questo punto l'incaricato del preside non si ferma ma continua a rivolgere la domanda ai restanti presenti. Infine si reca dal preside e comunica: «si, ce n'era uno».

Come si intuisce dalla storiella lo stesso compito può essere svolto in modi diversi. In alcuni casi si può dire che «funziona» nel senso che ottiene la risposta cercata, sebbene impieghi sforzi e tempo superflui. In altri casi si può dire che assolve al suo compito in modo efficiente, ovvero senza perdite inutili di risorse.

Sebbene non sia sempre facile trovare l'algoritmo più efficiente in assoluto, si può affermare che un algoritmo efficiente è migliore di uno solo efficace; quando lo spreco di risorse è evidente, l'algoritmo è considerato inefficiente e viene valutato meno del suo massimo possibile.

In particolare, quando si compiono algoritmi di ricerca, occorre soffermarsi a chiedersi se sia necessario esplorare tutto il vettore oppure se sia possibile anticipare la risposta.

Se, per esempio, a metà del vettore è possibile rispondere alla domanda posta e terminare la ricerca (come nella storiella citata prima) sarebbe inefficiente proseguire la ricerca per continuare a compiere verifiche superflue. In altri casi tuttavia è necessario esplorare tutto il vettore poiché non è possibile sapere se nella restante porzione vi siano dei dati che modificano la risposta finale.

UN ESERCIZIO PRIMA DI PROSEGUIRE

Per esercitarsi a capire si provi a rispondere se, ai seguenti problemi, è possibile dare una risposta prima di esplorare tutto il vettore oppure è necessario controllarlo tutto:

1.	Controllare se nel vettore c'è almeno uno zero	❑ Parte	Tutto
2.	Controllare se nel vettore sono tutti valori zero	❑ Parte	Tutto
3.	Ricerca del minimo	❑ Parte	Tutto
4.	Ricerca del primo valore dispari	❑ Parte	Tutto
5.	Verifica se un vettore è ordinato	❑ Parte	Tutto
6.	Controllare se nel vettore ci sono numeri negativi	❑ Parte	Tutto
7.	Controllare se in due vettori ci sono numeri in comune	❑ Parte	Tutto
8.	Contare i valori negativi in un vettore	❑ Parte	Tutto
9.	Calcolare la media dei valori pari del vettore	❑ Parte	Tutto
10.	Controllare se nel vettore ci sono numeri duplicati	❑ Parte	Tutto

ISTRUZIONE FOREACH

Abbiamo visto fino ad adesso tre dei quattro cicli disponibili in Visual C#. Vediamo ora la quarta istruzione iterativa: la **foreach**.

DESCRIZIONE E SCOPO DELL'ISTRUZIONE FOREACH

L'istruzione **foreach** ripete una istruzione (o un gruppo di istruzioni racchiuse in un blocco) per ciascuno degli elementi elencati in un vettore (o in generale matrice, collezione o insieme di oggetti). L'istruzione foreach viene utilizzata per scorrere l'insieme e leggere i dati desiderati.

La foreach ha il limite che non può essere utilizzata per modificare il contenuto dell'insieme; questo vincolo è posto con lo scopo di evitare effetti indesiderati.

L'esecuzione delle istruzioni incorporate viene ripetuta per ciascun elemento del vettore (o della matrice o dell'insieme). Quando l'iterazione è stata completata per tutti gli elementi dell'insieme, il controllo passa alla prima istruzione che segue il blocco foreach.

SINTASSI DELL'ISTRUZIONE FOREACH

La sintassi dell'istruzione **foreach** è la seguente:

```
foreach (Tipo variabile in Gruppo)
{
    Istruzioni1;
}
```

Se non si specificano le parentesi graffe del blocco, la foreach ammette una sola istruzione al suo interno. L'istruzione richiede le parentesi tonde (come il for); nelle parentesi si deve prima specificare il tipo della variabile che si intende usare nel ciclo; il tipo deve essere compatibile col tipo degli elementi del gruppo.

La parola chiave **in** serve per indicare la sorgente dei dati da analizzare che si scrive di seguito.

FUNZIONAMENTO DELL'ISTRUZIONE FOREACH

L'istruzione **foreach** crea una locazione temporanea dove ospitare, uno per volta (a turno), i dati del gruppo da scansionare. Se il gruppo non è vuoto la locazione assume il valore del primo elemento (ad indice zero, se indicizzato) e esegue l'istruzione racchiusa nel corpo del ciclo.

L'istruzione NON può modificare il dato del gruppo né la collezione stessa; questo significa che è vietato aggiungere o rimuovere elementi dal gruppo e che è vietato modificare il valore dell'elemento.

Dopo aver eseguito l'istruzione il ciclo ritorna alla variabile che prende il valore dell'elemento successivo e così via fino a quando vengono controllati tutti gli elementi del gruppo.

ESEMPIO 1. USO DELL'ISTRUZIONE FOREACH

```
//esempio di soluzione
label.Text = "";
int[] vettore=new int[6] {2, 3, 5, 7};
foreach (int i in vettore)
{
    label.Text += i + "; ";
}
```

- Pulisco la etichetta ponendo testo vuoto
- Dichiaro un vettore con 6 numeri dentro
- Esploro il vettore col foreach
- Per ciascun elemento nel vettore
- Lo visualizzo aggiungendolo al testo dell'etichetta

ESEMPIO 2. USO DELL'ISTRUZIONE FOREACH II

```
//esempio di soluzione
label.Text = "";
int[] vettore=new int[6] {2,5,9,3,4,0};
foreach (int i in vettore)
   if (i % 2 == 0)
      label.Text += i + "; ";
```

- Simile al precedente ma visualizza i soli numeri pari del vettore

ESEMPIO 3. USO DELL'ISTRUZIONE FOREACH III

```
listBox1.Items.Clear();
listBox2.Items.Clear();
for (int i=0; i<100; i++)
            listBox1.Items.Add(i);
foreach (object myElem in
listBox1.Items)
            listBox2.Items.Add(myElem);
```

- pulisce le due listBox
- poi prepara la prima con dei valori
- infine copia i valori dalla listBox1 alla listBox2

Si presti attenzione al fatto che se si esplorano collezioni come gli elenchi Items di ListBox e ComboBox occorre usare variabili di appoggio di tipo object, poiché il tentativo di usare un tipo intero (o altri tipi elementari) solleverebbe un errore.

RIEPILOGO DEGLI ALGORITMI DI BASE

RICERCA DEL VALORE MASSIMO IN UN VETTORE

```
int max = v[0];
for (int i = 0; i < v.Length; i++)
     if (max < v[i])
          max = v[i];
```

SOMMA DEI VALORI NUMERICI DI UN VETTORE

```
int som = 0;
for (int i = 0; i < v.Length; i++)
     som += v[i];
```

RICERCA E POSIZIONE DI UN VALORE IN UN VETTORE

```
//Ricerca di un valore e restituzione della sua posizione nel vettore
int valore = ...; //valore da cercare
int pos = -1; //posizione se lo si trova
for (int i = 0; i < v.Length; i++)
     if (valore == v[i])
     {
          pos = i;
          break;
     }
```

COPIA TRA VETTORI

```
int w = new int [v.Lenght]; //w ha stessa dimensione di v
for (int i = 0; i < v.Length; i++)
     w[i] = v[i];
```


CC BY NC SA

ORDINAMENTO INGENUO DI UN VETTORE

```
for (int k = 0; k < pippo.Length -1; k++)
   for (int i = 0; i < pippo.Length -1; i++)
      if ( pippo[i] > pippo[i + 1] )
      {
          int tmp = pippo[i];
          pippo[i] = pippo[i + 1];
          pippo[i + 1] = tmp;
      }
```

ALGEBRA DEI VETTORI

Sui vettori sono state costruite algebre potenti e complicate. Vedremo in questa dispensa alcuni degli algoritmi più noti, senza entrare nel merito dei significati geometrici, fisici e matematici delle operazioni realizzate.

SOMMA VETTORIALE

La somma di due vettori, di **stessa dimensione**, v e w è un vettore u = v + w i cui elementi si ottengono sommano i rispettivi elementi (stessa posizione);

un esempio è:

v →	**3**	**7**	**2**	**0**	**1**	**4**	**5**
+							
w →	**0**	**1**	**3**	**4**	**8**	**1**	**2**
=							
u →	**3**	**8**	**5**	**4**	**9**	**5**	**7**

Dalla definizione si deducono le seguenti proprietà:

la somma è associativa, commutativa e possiede l'elemento neutro che è il vettore nullo (vettore che contiene zero in ogni elemento); inoltre ogni elemento ha un opposto (il negativo).

PRODOTTO DI UNO SCALARE PER UN VETTORE

Il prodotto di un valore numerico r (reale) per un vettore v, è un vettore u = r*v i cui elementi si ottengono moltiplicando ciascun elemento per il numero;

un esempio è:

r → 4							
x							
v →	**3**	**-1**	**5**	**0**	**1**	**-3**	**2**
=							
u →	**12**	**-4**	**20**	**0**	**4**	**-12**	**8**

Il prodotto scalare per vettore è associativo, gode di proprietà distributive; vale la regola 1v=v; vale la regola 0v=O (dove O è il vettore nullo).

PRODOTTO SCALARE DI VETTORI

Il prodotto scalare euclideo di due vettori, di **stessa dimensione**, v e w è il numero r ottenuto dalla sommatoria dei prodotti delle celle con stessa posizione, ovvero:

r = v[0]*w[0] + v[1]*w[1] + … + v[N-1]*w[N-1];

un esempio è:

v →	3	7	2	0	1	4	5
x							
w →	0	1	3	4	8	1	2
=							

v*w = 3x0 + 7x1 + 2x3 + 0x4 + 1x8 + 4x1 + 5x2 = 0 + 7 + 6 + 0 + 8 + 4 + 10 = **35**

Il prodotto scalare tra due vettori viene indicato usualmente con uno dei simboli seguenti: $\langle \mathbf{v}, \mathbf{w} \rangle$ $\mathbf{v} \cdot \mathbf{w}$ $\mathbf{v}\,\mathbf{w}$

NORMA DI UN VETTORE

La norma euclidea di un vettore v è il numero ottenuto dalla radice quadrata del prodotto scalare per sé stesso. Ovvero:

$\|v\|$ = v[0]*v[0] + v[1]*v[1] + … + v[N-1]*v[N-1];

un esempio è mostrato di lato:

v →	1	2	0	2	1	9	3
=							

$\|\mathbf{v}\|$ =

= √(1x1 + 2x2 + 0x0 + 2x2 + 1x1 + 9x9 + 3x3) =

= √(1 + 4 + 0 + 4 + 1 + 81 + 9) = √(100) = **10**

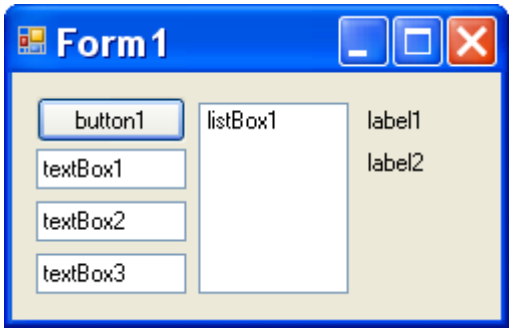
Questa quantità è considerata esser eguale alla lunghezza (fisica o geometrica) del vettore.

ESERCIZI

ESERCIZI SUI VETTORI

Prima leggi gli esercizi elencati di seguito. Prova a svolgerli da solo uno alla volta. Dopo verifica la tua soluzione con quella mostrata alla fine del documento.

Per risolvere gli esercizi è possibile creare una applicazione simile alla seguente figura e associare gli esercizi al pulsante



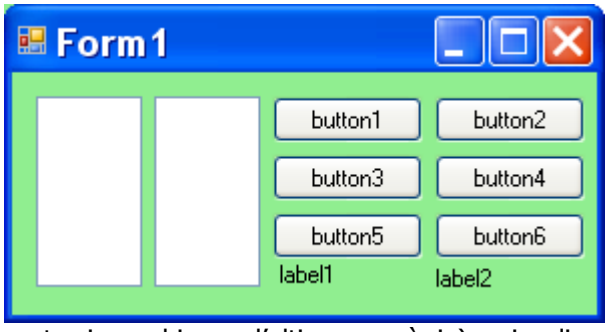
Esercizio 1) USO BANALE DEL VETTORE

Dichiarare un vettore di interi.

Il pulsante deve allocare memoria per il vettore con un numero di elementi specificato nella textBox1, poi inizializzarlo con valori casuali interi compresi tra gli estremi indicati dalle textBox2 e textBox3. Infine visualizzare nel listBox1 il vettore e nelle due etichette i valori minimo e massimo.

Esercizio 2) MANIPOLAZIONE DEL VETTORE

Dichiarare un vettore di interi.



Pulsante1 : allocare memoria per il vettore con un numero di elementi specificato nella textBox1, poi inizializzarlo con valori casuali interi compresi tra 10 e 99 estremi inclusi.

Pulsante2 : Visualizzare il vettore nella listBox1.

Pulsante3 : invertire rovesciare il vettore (il primo elemento si scambia con l'ultimo e così via) e visualizzare il vettore risultante nella listBox2. NON usare un vettore di appoggio, ma solo una variabile intera.

Esercizio 3) NUMERI FRAZIONARI RELATIVI

Dichiarare un vettore di decimali.

Un primo pulsante deve allocare memoria per il vettore, per 100 celle, con valori compresi tra -1 e +1, estremi esclusi. Si osservi che si devono generare numeri con la virgola come per esempio 0,975 oppure -0,853 (negativo).

Un secondo pulsante deve calcolare la media dei valori negativi, di quelli positivi deve contare gli zeri. I tre valori vanno visualizzati in un listBox1 preventivamente pulito.

Esercizio 4) PRODOTTO SCALARE

Dichiarare due vettori di decimali.

Un primo pulsante deve allocare memoria per i due vettori, per 100 celle ciascuno.

Un secondo pulsante deve calcolare il prodotto scalare dei vettori e mostrarlo nella label1.

Esercizio 5) FUSIONE DI VETTORI

Dichiarare due vettori di interi.

Un primo pulsante deve allocare memoria per i due vettori, entrambi con un numero di elementi specificato nella textBox1.

Un secondo pulsante deve invece fondere i due vettori già inizializzati, ottenendone un terzo che verrà visualizzato in un listBox1.

Esercizio 6) RICERCA DI VALORI COMUNI

Dichiarare due vettori di interi.

Un primo pulsante deve allocare memoria per i due vettori rispettivamente con un numero di elementi specificato nelle textBox1 e texBox2, e deve infine riempire i vettori con numeri casuali compresi tra 0 e 99, estremi inclusi.

Un secondo pulsante deve verificare se esiste almeno un numero che compare in entrambi i vettori; se ne trova uno deve visualizzare in una label1 il valore e nelle label2 e label3 rispettivamente gli indici dove si trova il valore nei due vettori.

Esercizio 7) RICERCA DI VALORI DUPLICATI

Dichiarare un vettore di interi.

Un primo pulsante deve allocare memoria per il vettore con 100 celle e deve infine riempirlo con numeri casuali compresi tra -25 e +25, estremi inclusi.

Un secondo pulsante deve verificare se esiste almeno un numero che compare due volte nel vettore (ovviamente in posizioni diverse); se ne trova uno deve visualizzare in una label1 il valore e nelle label2 e label3 rispettivamente gli indici dove si trova il valore nel vettore.

SOMMARIO