Fondamenti di Informatica T-1
Modulo 2

Obiettivi di questa esercitazione

1. Array e funzioni
2. Array e funzioni ricorsive
3. Array e confronto di array

Esercizio 1

(array e funzioni)

- Creare un programma che legga da input un numero non noto a priori di interi (al più 10) terminati da 0. Tale sequenza può eventualmente contenere numeri ripetuti. A tal fine, si realizzi una funzione apposita che riceva in ingresso un vettore (vuoto) e la sua dimensione fisica, e restituisca la dimensione logica.

- Si chieda poi l'inserimento di un valore k da cercare nell'array.

- Si realizzi una procedura che stampi a video tutti gli indici relativi alle posizioni in cui il numero k è presente nell'array. La procedura riceverà quindi come parametri l'array, la sua dimensione logica e il valore k.

Esercizio 1 - Soluzione

(array e funzioni)

```
#include <stdio.h>
#include <stdlib.h>
#define DIM 10

int leggi(int vet[], int dim) {
    int size = 0, num;
    do {
          printf("Inserisci un numero: ");
          scanf("%d", &num);
          if (num!=0 && size<dim) {
                    vet[size] = num;
                    size++;
          }
    } while (num!=0 && size<dim);
    return size;
}

    …
```

Esercizio 1 - Soluzione

(array e funzioni)

```
void search(int values[], int dim, int k) {
   int i;
   for (i=0; i<dim; i++)
        if (values[i] == k)
               printf("Trovato il valore %d all'indice %d\n ", k, i);
}

int main(void) {
   int num, size, i, k;
   int values[DIM];

   size = leggi(values, DIM);

   printf("Inserisci valore da cercare: ");
   scanf("%d", &k);
   search(values, size, k);

   system("PAUSE");
   return (0);
}
```

Esercizio 2

(array e funzioni)

- Realizzare una funzione che riceva in ingresso un array di interi e la sua dimensione, un elemento da cercare ed un intero passato per riferimento.
- La funzione deve restituire un valore interpretabile come “vero” se l’elemento è presente nell’array. Inoltre, tramite l’intero passato per riferimento, la funzione deve restituire anche la posizione dell’elemento nell’array
- Realizzare anche un main di esempio

Esercizio 2 - Soluzione

(array e funzioni)

```
int trovaPos(int vet[], int dim, int el, int *pos) {

   int trovato, i;

   trovato = 0;
   for (i=0; i<dim && trovato==0; i++) {
       if (vet[i] == el) {
              trovato = 1;
              *pos = i;
       }
   }
   return trovato;
}
```

Esercizio 3
(array e funzioni)

- Creare un programma che legga da input due sequenze di interi, di lunghezza non nota a priori (al più 10), e terminate da 0. A tal scopo, si realizzi una apposita funzione.
- Si realizzi un main che invoca le funzioni, e che stampi a video tutti gli elementi del primo vettore che compaiono nel secondo vettore nella stessa posizione (cioè con lo stesso indice). Si supponga per semplicità che le due sequenze abbiano la stessa lunghezza.

Esercizio 3 - Soluzione

(array e funzioni)

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

int leggi(int vet[], int dim) {
    int i, num;

    i=0;
    do {
          printf("Inserisci numero: ");
          scanf("%d", &num);
          if (num != 0 && i<dim) {
                    vet[i] = num;
                    i++;
          }
    } while (num!=0 && i<dim);
    return i;
}

…
```

Esercizio 3 - Soluzione
(array e funzioni)

```
int main(void) {

    int v1[10], v2[10], i;
    int dim_v1, dim_v2;

    dim_v1 = leggi(v1, 10);
    dim_v2 = leggi(v2, 10);

    for (i=0; i<dim_v1; i++)
         if ( v1[i] == v2[i])
                  printf("%d ", v1[i]);

    system("PAUSE");
    return (0);
}
```

Esercizio 4
(array e funzioni)

- Realizzare una procedura che, ricevuti in ingresso un vettore di interi e la sua dimensione, e due interi passati per riferimento di nome “pari” e “dispari”, restituisca il numero di interi pari e di interi dispari presenti nell’array.
- Si realizzi un main che, utilizzando una appropriata funzione, legga dall’utente una sequenza di al più 10 numeri (eventualmente terminati da zero), e utilizzando la procedura di cui al punto precedente, stampi a video quanti numeri pari e dispari sono stati inseriti.

Esercizio 4 - Soluzione

(array e funzioni)

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

int leggi(int vet[], int dim) {
    int i, num;
    i=0;
    do {
          printf("Inserisci numero: ");
          scanf("%d", &num);
          if (num != 0 && i<dim) {
                    vet[i] = num;
                    i++;
          }
    } while (num!=0 && i<dim);
    return i;
}

void contaPariDisp(int vet[], int dim, int * pari, int * disp) {
    int i;

    *pari = 0;
    *disp = 0;
    for (i=0; i<dim; i++) {
          if ( (vet[i]%2)==0)
                    (*pari)++;
          else
                    (*disp)++;
    }
}
```

Esercizio 4 - Soluzioni

(array e funzioni)

```
…

int main(void)
{

   int vet[10], pari, disp, dim;

   dim = leggi(vet, 10);

   contaPariDisp(vet, dim, &pari, &disp);

   printf ("l'array contiene %d numeri pari e %d dispari", pari, disp);


   system("PAUSE");
   return (0);
}
```

Esercizio 5
(array e funzioni)

- Creare un programma che legga da input due sequenze di interi, di lunghezza non nota a priori (al più 10), e terminate da 0. A tal fine, si realizzi una funzione apposita che riceva come parametri un vettore vuoto (da riempire) e la sua dimensione fisica, e restituisca la dimensione logica.
- Per semplicità, si ipotizzi che ogni sequenza non contenga elementi ripetuti

- Il programma poi memorizzi in un terzo vettore tutti gli elementi che compaiono in entrambi gli array iniziali (intersezione), e lo si stampi a video

Esercizio 5 - Soluzione

(array e funzioni)

```
#include <stdio.h>
#include <stdlib.h>
#define DIM 10

int leggi(int vet[], int dim) {
    int size = 0, num;
    do {
          printf("Inserisci un numero: ");
          scanf("%d", &num);
          if (num!=0 && size<dim) {
                  vet[size] = num;
                  size++;
          }
    } while (num!=0 && size<dim);
    return size;
}

int main(void) {
    int num, size1, size2, size3, i, j, trovato;
    int values1[DIM], values2[DIM], intersez[DIM];
    size1 = 0; size2 = 0; size3 = 0;
    size1 = leggi(values1, DIM);
    size2 = leggi(values2, DIM);
    …
```

Esercizio 5 - Soluzione

(array e funzioni)

```
    …
    for (i=0; i<size1; i++) {
         trovato = 0;
         for (j=0; j<size2 && !trovato; j++)
                 if (values1[i] == values2[j])
                         trovato = 1;
         if (trovato) {
                 intersez[size3] = values1[i];
                 size3++;
         }
    }
    for (i=0; i<size3; i++)
         printf("Valore comune: %d\n", intersez[i]);

    system("PAUSE");
    return (0);
}
```

Esercizio 6
(array e funzioni)

- Creare un programma che legga da input una sequenza di interi, di lunghezza non nota a priori (al più 10), e terminata da 0. A tal scopo, si realizzi una funzione che riceva come parametri di ingresso un vettore e la sua dimensione fisica, e restituisca la dimensione logica del vettore. Tale funzione si deve fare carico della fase di lettura e riempimento dell'array.
- La sequenza può contenere elementi ripetuti (anche più volte).

- Si realizzi una funzione che, ricevuti in ingresso il primo vettore con la sua dimensione logica, ed un secondo vettore con la sua dimensione fisica, memorizzi nel secondo vettore tutti gli elementi del primo, ma senza ripetizioni. La funzione restituisca la dimensione logica del secondo vettore.

- Si realizzi un main che invoca le funzioni, e che stampi a video l'elenco degli elementi non ripetuti

Esercizio 6 - Soluzione

(array e funzioni)

```
#include <stdio.h>
#include <stdlib.h>
#define DIM 10

int leggi(int vet[], int dim) {
    int size = 0, num;
    do {
          printf("Inserisci un numero: ");
          scanf("%d", &num);
          if (num!=0 && size<dim) {
                    vet[size] = num;
                    size++;
          }
    } while (num!=0 && size<dim);
    return size;
}


    …
```

Esercizio 6 - Soluzione

(array e funzioni)

```
int eliminaRipetuti(int[] values, int dim_v, int[] single, int dim_s) {
    int size_s = 0;
    int i, j, trovato;

    for (i=0; i<dim_v && size_s < dim_s; i++) {
         trovato = 0;
         for (j=0; j<size_s && !trovato; j++) {
                  if (values[i] == single[j])
                  trovato = 1;
         }
         if (!trovato) {
                  single[size_s] = values[i];
                  size_s++;
         }
    }
    return size_s;
}
```

Esercizio 6 - Soluzione

(array e funzioni)

```
int main(void) {
    int num, size_v, size_s, i, j, trovato;
    int values[DIM], single[DIM];

    size_v = leggi(values, DIM);

    size_s = eliminaRipetuti(values, size_v, single, DIM);

    for (i=0; i<size_s; i++)
         printf("%d ", single[i]);

    system("PAUSE");
    return 0;
}
```

Esercizio 7

(array e funzioni)

- Creare un programma che legga da input una sequenza di interi, di lunghezza non nota a priori (al più 10), e terminata da 0. A tal scopo, si realizzi una funzione che riceva come parametri di ingresso un vettore e la sua dimensione fisica, e restituisca la dimensione logica del vettore. Tale funzione si deve fare carico della fase di lettura e riempimento dell'array. La sequenza può contenere elementi ripetuti (anche più volte).
- Si realizzi una funzione che, ricevuti un array e la sua dimensione, ed un elemento da cercare, *restituisca il valore -1 se l'elemento non è presente nell'array; altrimenti restituisca il primo indice in cui è presente l'elemento cercato*.
- Si realizzi un main che invoca le funzioni, e che stampi a video l'elenco degli elementi che compaiono più volte nel vettore

Esercizio 7 - Soluzione

(array e funzioni)

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

int leggi(int vet[], int dim) {
    int i, num;

    i=0;
    do {
          printf("Inserisci numero: ");
          scanf("%d", &num);
          if (num != 0 && i<dim) {
                    vet[i] = num;
                    i++;
          }
    } while (num!=0 && i<dim);
    return i;
}

…
```

Esercizio 7 - Soluzione

(array e funzioni)

```
…

int trovaPos(int vet[], int dim, int el) {
    int trovato, i;

    trovato = -1;
    for (i=0; i<dim && trovato<0; i++) {
          if (vet[i] == el)
                    trovato = i;
    }
    return trovato;
}

…
```

Esercizio 7 - Soluzione

(array e funzioni)

```
int main(void)
{

   int v[10], dim, i;

   dim = leggi(v, 10);
   for (i=0; i<dim; i++)
        if ( trovaPos(&v[i+1], dim-i-1, v[i]) >= 0 )
                if ( trovaPos(v, i, v[i]) < 0)
                        printf("%d ", v[i]);


   system("PAUSE");

   return (0);
}
```

Esercizio 8

(array e funzioni)

- Creare un programma che legga da input due sequenze di interi, di lunghezza non nota a priori (al più 10), e terminate da 0. A tal scopo, si realizzi una apposita funzione.
- Si realizzi una funzione che, ricevuti un array e la sua dimensione, ed un elemento da cercare, *restituisca il valore -1 se l'elemento non è presente nell'array; altrimenti restituisca il primo indice in cui è presente l'elemento cercato*.
- Si realizzi un main che invoca le funzioni, e che stampi a video tutti gli elementi del primo vettore che NON compaiono nel secondo.

Esercizio 8 - Soluzione

(array e funzioni)

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

int leggi(int vet[], int dim) {
    int i, num;

    i=0;
    do {
          printf("Inserisci numero: ");
          scanf("%d", &num);
          if (num != 0 && i<dim) {
                    vet[i] = num;
                    i++;
          }
    } while (num!=0 && i<dim);
    return i;
}

…
```

Esercizio 8 - Soluzione

(array e funzioni)

```
…

int trovaPos(int vet[], int dim, int el) {
    int trovato, i;

    trovato = -1;
    for (i=0; i<dim && trovato<0; i++) {
          if (vet[i] == el)
                    trovato = i;
    }
    return trovato;
}

…
```

Esercizio 8 - Soluzione

(array e funzioni)

```
…

int main(void)
{
    int v1[10], v2[10], i;
    int dim_v1, dim_v2;

    dim_v1 = leggi(v1, 10);
    dim_v2 = leggi(v2, 10);

    for (i=0; i<dim_v1; i++)
         if (trovaPos(v2, dim_v2, v1[i]) < 0)
                  printf("%d ", v1[i]);


    system("PAUSE");

    return (0);
}
```

Esercizio 9

(array e funzioni)

- Creare un programma che legga da input due sequenze di interi, di lunghezza non nota a priori (al più 10), e terminate da 0. A tal scopo, si realizzi una apposita funzione.
- Si ipotizzi che le sequenze di numeri inseriti siano ordinate in maniera crescente.
- Si realizzi un main che invochi la funzione per leggere due sequenze, e che stampi a video, in ordine crescente, tutti gli elementi di entrambi i vettori. Ad esempio, con v1={1,3,5,7} e v2={2,4,6,8} il programma deve stampare: 1,2,3,4,5,6,7,8

Esercizio 9 - Soluzione

(array e funzioni)

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

int leggi(int vet[], int dim) {
    int i, num;

    i=0;
    do {
          printf("Inserisci numero: ");
          scanf("%d", &num);
          if (num != 0) {
                    vet[i] = num;
                    i++;
          }
    } while (num!=0 && i<dim);
    return i;
}

…
```

Esercizio 9 - Soluzione

(array e funzioni)

```
int main(void) {
    int v1[10], v2[10], i, j;
    int dim_v1, dim_v2;

    dim_v1 = leggi(v1, 10);
    dim_v2 = leggi(v2, 10);

    i = 0;
    j = 0;
    while (i<dim_v1 && j<dim_v2) {
          if (v1[i] < v2[j]) {
                printf("%d ", v1[i]);
                i++;
          }
          else {
                printf("%d ", v2[j]);
                j++;
          }
    }
    while (i<dim_v1) {
          printf("%d ", v1[i]);
          i++;
    }
    while (j<dim_v2) {
          printf("%d ", v2[j]);
          j++;
    }
    system("PAUSE"); return (0); }
```

Esercizio 10

(array e funzioni)

- Creare un programma che legga da input una sequenza di interi, di lunghezza non nota a priori (al più 10), e terminata da 0. A tal scopo, si realizzi una apposita funzione.
- Si realizzi una funzione che, ricevuti un array e la sua dimensione, ed un elemento da cercare, *restituisca il valore -1 se l'elemento non è presente nell'array; altrimenti restituisca il primo indice in cui è presente l'elemento cercato*.
- Si realizzi un main che invoca le funzioni, e che stampi a video tutti gli elementi del vettore che compaiono solo ed esattamente 2 volte nel vettore.

Esercizio 10 - Soluzione

(array e funzioni)

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

int leggi(int vet[], int dim) {
    int i, num;

    i=0;
    do {
          printf("Inserisci numero: ");
          scanf("%d", &num);
          if (num != 0 && i<dim) {
                    vet[i] = num;
                    i++;
          }
    } while (num!=0 && i<dim);
    return i;
}

…
```

Esercizio 10 - Soluzione

(array e funzioni)

```
…

int trovaPos(int vet[], int dim, int el) {
    int trovato, i;

    trovato = -1;
    for (i=0; i<dim && trovato<0; i++) {
          if (vet[i] == el)
                    trovato = i;
    }
    return trovato;
}

…
```

Esercizio 10 - Soluzione

(array e funzioni)

```
int main(void) {
    int v[10], dim, i, count, temp_dim, temp_pos, offset;

    dim = leggi(v, 10);

    for (i=0; i<dim; i++) {
          count = 0;
          temp_dim = dim;
          offset = 0;
          do {
                    temp_pos = trovaPos(&v[offset], temp_dim, v[i]);
                    if (temp_pos >= 0) {
                              count++;
                              temp_dim = temp_dim - temp_pos - 1;
                              offset = offset + temp_pos + 1;
                    }
          } while (temp_pos >= 0);
          if (count == 2)
                    printf("L'elemento %d compare 2 volte\n", v[i]);

    }
    system("PAUSE"); return (0); }
```

Esercizio 11

(array e funzioni)

- Si vuole realizzare una funzione che, dati un array di valori interi, ordinati non ripetuti, e due valori estremi, restituisca il sotto-array compreso tra i due estremi.
- Tale funzione quindi riceverà in ingresso un vettore di interi e la sua dimensione; due interi di nome “first” e “last”; un intero dim passato per riferimento. La funzione dovrà restituire un puntatore all’elemento dell’array pari a first, se presente, e tramite dim la dimensione logica del sotto-array
- Ad esempio, se invocata con v={1,2,3,5,6,8,9}, first=3, last=8, la funzione deve restituire il puntatore all’elemento all’indice 2 (&v[2]), e dimensione 4.

Esercizio 11 - Soluzione

(array e funzioni)

```
int * select(int v[], int length, int first, int last, int * dim) {
   int i;
   int * result;

   i=0;
   while (i<length && first>v[i])
        i++;
   result = &(v[i]);

   *dim = 0;
   while (i<length && last>=v[i]) {
        i++;
        *dim = *dim + 1;
   }
   return result;
}
```

…

Esercizio 11 - Soluzione

(array e funzioni)

```
int main(void)
{
   int v[10], dim_v;
   int * v2;
   int dim_v2;
   int first, last, i;

   dim_v = leggi(v, 10);
   printf("Inserisci i due estremi: ");
   scanf("%d%d", &first, &last);
   v2 = select(v, dim_v, first, last, &dim_v2);
   for (i=0; i<dim_v2; i++)
        printf("%d ", v2[i]);

   system("PAUSE");

   return (0);
}
```

Esercizio 12

(array e funzioni ricorsive)

Scrivere una procedura ricorsiva:

void print(int list[], int length)

che stampi, *ricorsivamente*, tutti i numeri contenuti nell'array **list**.

Esercizio 12 - Soluzione

(array e funzioni ricorsive)

```
void print (int list[], int length) {
   if (length != 0)
   {   print(list, length-1);
       printf("%d\n", list[length-1]);
   }
}
```

Esercizio 13

(array e funzioni ricorsive)

Scrivere una funzione ricorsiva che, ricevuto in ingresso un array di interi, esegua la somma degli interi in posizione con indice dispari.

Esercizio 13 - Soluzione

(array e funzioni ricorsive)

```
int sumOdd2(int list[], int length, int pos) {
   if (pos >=length)
       return 0;
   else
       return list[pos] + sumOdd2(list, length, pos+2);
}

int sumOdd(int list[], int length) {
   return sumOdd2(list, length, 1);
}
```

Esercizio 13 – Soluzione - Variante
(array e funzioni ricorsive)

```
int sumOdd(int list[], int length) {
   if (length >0) {
        if ( ((length-1)%2) == 0 )
               return sumOdd(list, length-1);
        else
               return list[length-1] + sumOdd(list, length-2);
   }
   else
        return 0;
}
```

Esercizio 14

(array e funzioni ricorsive)

Si definisca una procedura ricorsiva:

```
void somme2(int l1[], int length)
```

che, ricevuto in ingresso un array l1 di interi, stampi a video gli interi dell'array di ingresso il cui valore è uguale alla somma dei due interi seguenti nell'array (a tal fine, gli ultimi due numeri di un'array sono automaticamente esclusi).

Ad esempio, se invocata con l1 = [5, 6, 4, 2, 1, 1, 3, 1], la procedura deve stampare [6, 2]. Infatti, considerando il primo valore (5): i due valori successivi sono 6 e 4, e la loro somma vale 10; quindi 5 è scartato. Per il secondo valore, pari a 6, la somma dei due valori successivi è proprio 6 e quindi il valore viene selezionato per la stampa.

Esercizio 14 - Soluzione

(array e funzioni ricorsive)

```
void somme2(int l1[], int length) {
   if (length<3)
       return;
   else {
       if (l1[0] == l1[1] + l1[2]) {
             printf("%d, ", l1[0]);
       }
       somme2(&(l1[1]), length-1);
   }
}
```

Esercizio 15
(array e funzioni)

Test di uguaglianza fra vettori, elemento per elemento

- Creare una funzione che, dati in input due vettori e le rispettive lunghezze, determini se i due vettori sono uguali
- IPOTESI: l'uguaglianza va testata in maniera "ordinata", ovvero elemento per elemento

Esercizio 15
(array e funzioni)

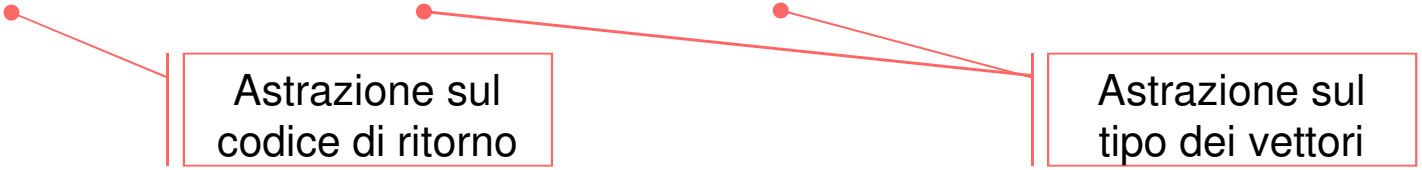
Linee guida per la soluzione

- Cerchiamo di astrarre il più possibile sul tipo dei vettori
  - Uso di costanti simboliche
  - Incapsulamento del test di uguaglianza fra due elementi in una funzione a parte
    - Questa è l'unica funzione che deve conoscere quali sono i tipi!
- Restituizione di codici differenziati
  - Uso di costanti simboliche
- Un occhio all'efficienza
  - ***Cerchiamo di effettuare il minor numero di cicli possibile***

Esercizio 15 - Suggerimenti
(array e funzioni)

- PRIMO PASSO: dichiarazione delle funzioni

```
RESULT compareTo(TYPE v1[], TYPE v2[], int dim1, int dim2)
```



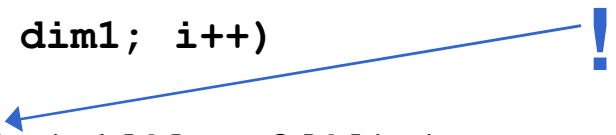

```
BOOLEAN equals(TYPE el1, TYPE el2)
```

- SECONDO PASSO: pseudocodice

```
se dim1 != dim2 i vettori sono DIFFERENTI
per i da 0 alla lunghezza dei vettori
{
    se i-mo elemento di v1 diverso
    rispetto a i-mo elemento di v2,
        i vettori sono DIFFERENTI
}
i vettori sono UGUALI
```

Esercizio 15 - Soluzione
(array e funzioni)

```
#define RESULT int
#define DIFFERENT_LENGTH -1
#define EQUAL 0
#define DIFFERENT 1
RESULT compareTo1(TYPE v1[], TYPE v2[], int dim1, int dim2)
{
   int i;
   if(dim1 != dim2)
        return DIFFERENT_LENGTH;
   for(i = 0; i < dim1; i++)
   {
        if( !equals(v1[i], v2[i]) )
                return DIFFERENT;
   }
   return EQUAL;
}
```



Esercizio 15
(array e funzioni)

- L'uguaglianza fra elementi deve ovviamente conoscere il loro tipo…
- Es: per vettori di interi

```
#define TYPE int
#define BOOLEAN int
#define TRUE 1
#define FALSE 0

BOOLEAN equals(TYPE el1, TYPE el2)
{
   return(el1 == el2); //uguaglianza dipendente dal tipo
}
```

Esercizio 16
(array e funzioni)

Test di uguaglianza fra vettori con elementi non ripetuti

- Questa volta il test deve verificare che i vettori contengano gli stessi elementi, NON NECESSARIAMENTE NELLO STESSO ORDINE
- Ipotesi semplificativa: i vettori non hanno elementi ripetuti

Esercizio 16
(array e funzioni)

- Che cosa cambia rispetto al test precedente?
  - Non dobbiamo più controllare semplicemente se tutti gli elementi di indice uguale sono uguali
  - Dobbiamo piuttosto verificare che ogni elemento del primo vettore sia contenuto nel secondo
  - Questo è sufficiente?
    - Ricordarsi delle ipotesi!

Esercizio 16 - Soluzione
(array e funzioni)

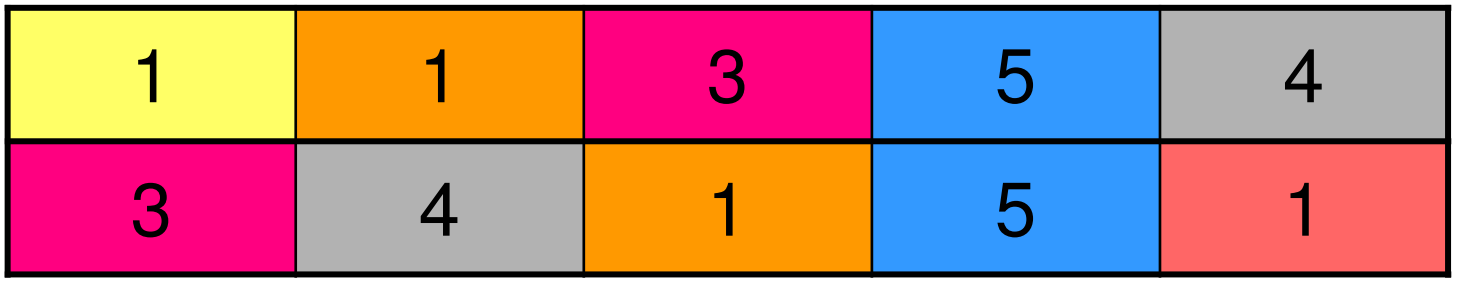
```
RESULT compareTo2(TYPE v1[], TYPE v2[], int dim1, int dim2)
{
   int i, j;
   BOOLEAN currentEquality = TRUE;
   if(dim1 != dim2)
        return DIFFERENT_LENGTH;
   for(i = 0; i < dim1 && currentEquality; i++)
   {                                          contains(v2, v1[i])
        currentEquality = FALSE;
        for(j = 0; j < dim2 && !currentEquality; j++)
        {
                currentEquality = equals(v1[i], v2[j]);
        }
        if(!currentEquality)
                return DIFFERENT;
   }
   if(currentEquality)
        return EQUAL;
   else
        return DIFFERENT;
}
```

Esercizio 17
(array e funzioni)

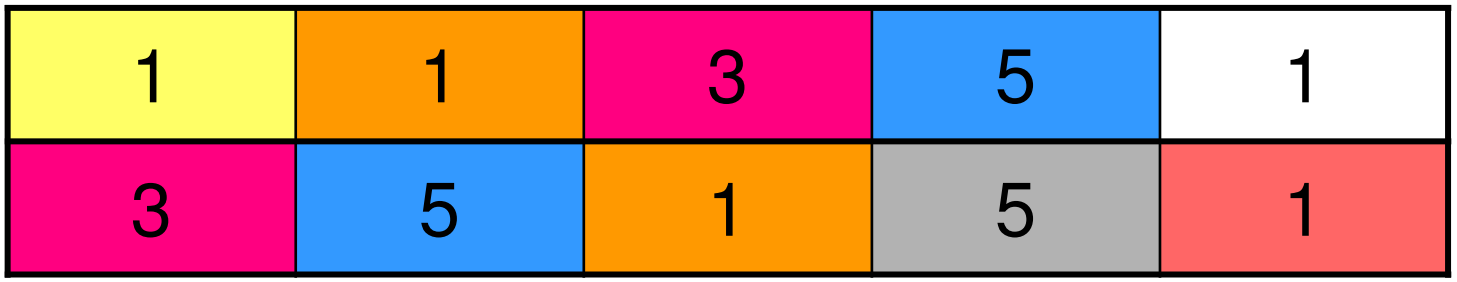
Test di uguaglianza fra vettori con elementi ripetuti

- Ora rimuoviamo anche l'ipotesi sulla possibilità di avere elementi ripetuti
- Che cosa dobbiamo modificare della precedente funzione?
  - Cerchiamo, quando è sensato ☹, di riutilizzare ciò che abbiamo già realizzato

Esercizio 17
(array e funzioni)



SI



NO

- *Occorre controllare che ci sia una corrispondenza uno a uno fra gli elementi*
- *È necessaria una struttura dati di appoggio!*

Esercizio 17 - Soluzione
(array e funzioni)

Per risolvere il problema della corrispondenza uno a uno

- Definiamo un ***vettore di booleani***, con dimensione logica pari a quella dei vettori in esame
- Inizializziamo ogni elemento del vettore a FALSE
- L'elemento i-mo di tale vettore indica se l'elemento del secondo vettore in esame ***è già stato utilizzato*** per un confronto di successo
- Quindi il test di appartenenza di un elemento del primo vettore nel secondo deve considerare unicamente gli ***elementi non ancora utilizzati per confronti di successo***
  - Ovvero quelli per cui il corrispondente valore booleano è ancora FALSE

Esercizio 17 - Soluzione
(array e funzioni)

```
#define MAX_DIM 50
RESULT compareTo3(TYPE v1[], TYPE v2[], int dim1, int dim2)
{
   BOOLEAN checked[MAX_DIM];
   int i, j;
   BOOLEAN currentEquality = TRUE;
   if(dim1 != dim2)
        return DIFFERENT_LENGTH;
   for(i = 0; i < dim1; i++) //uso la dimensione logica
        checked[i] = FALSE;
   …
```

Esercizio 17 - Soluzione
(array e funzioni)

```
for(i = 0; i < dim1 && currentEquality; i++)
{
        currentEquality = FALSE;
        for(j = 0; j < dim2 && !currentEquality; j++)
        {
                if(!checked[j])
                {
                        currentEquality = equals(v1[i], v2[j]);
                        if(currentEquality)
                        {
                                checked[j] = TRUE;
                        }
                }
        }
}
if(currentEquality)
        return EQUAL;
else
        return DIFFERENT;
}
```

Uso solo gli elementi non ancora utilizzati per un confronto di successo

v2[j] è stato utilizzato per un confronto di successo

➔ Devo aggiornare il vettore di booleani