Laboratorio di Calcolatori 1

Corso di Laurea in Fisica
A.A. 2007/2008

Dott.Davide Di Ruscio

Dipartimento di Informatica

Università degli Studi di L'Aquila

Lezione del 17/04/08

Nota

Questi lucidi sono tratti dal materiale distribuito dalla McGraw-Hill e basati su del materiale fornito dal Prof. Flammini Michele

Sommario (II parte)

Il Linguaggio C

- Caratteristiche generali
- Un linguaggio C semplificato ed esempi di semplici programmi
- Struttura di un programma C
- Direttive del pre-processore
- Parte dichiarativa:
  - tipi
  - definizioni di tipi
  - definizioni di variabili
- Parte esecutiiva
  - istruzione di assegnamento
  - istruzioni (funzioni) di input-output
  - istruzioni di selezione
  - istruzioni iterative
- Vettori mono e multidimensionali
- Funzioni e procedure
- File
- Allocazione dinamica di memoria
- Suddivisione dei programmi in piu' file e compilazione separata
- Algoritmi elementari
  - ricerca sequenziale e binaria
  - ordinamento di un vettore: per selezione, per inserimento, per fusione e a bolle
- Aspetti avanzati di programmazione
  - ricorsione
  - strutture dati dinamiche

RIFERIMENTI

Ceri, Mandrioli, Sbattella

Informatica arte e mestiere

McGraw-Hill

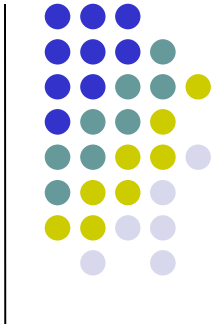
Ancora sui tipi: definizione di tipo

- Dichiarazione di un nuovo tipo
  - Effettuata normalmente all'inizio del programma
  - Si utilizza il costruttore di tipo **typedef,** eventualmente con altri costruttori tipi **struct, array**

- Il costruttore **typedef** consente di dichiarare nuovi tipi elencando nell'ordine:
  - La rappresentazione del nuovo tipo
  - Il nome del nuovo tipo
  - Termina sempre con un ;

- L'utente può definire nuovi tipi mediante:
  - Ridefinizione
  - Applicazione del costrutto di enumerazione esplicita

Ridefinizione di tipi



- Sintattica:

```
typedef        TipoEsistente  NuovoTipo;
```

- Dove:
  - TipoEsistente può essere sia un tipo *built-in* (predefinito), sia un tipo precedentemente definito
  - NuovoTipo è nome che identifica il nuovo tipo

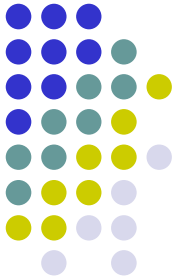
- Esempi:

  -
```
typedef    int    anno;
typedef    char   tipo1;
typedef    anno   tipo2;
typedef    tipo2  tipo3;
```

Una volta definito e identificato un nuovo tipo ogni variabile può essere dichiarata di quel tipo come di ogni altro tipo già esistente:

```
char   x;
anno   y;
```

Enumerazione esplicita di valori



- Un nuovo tipo dichiarato con la parola chiave **typedef,** seguito da
  - la parola chiave **enum**
  - la lista dei possibili valori fra parentisi graffe (separati da virgole)
  - il nome del nuovo tipo
  - termina con un punto e virgola

- Esempio:

```
typedef enum    {lun, mar, mer, gio, ven, sab, dom}  GiornoDellaSettimana;
typedef enum    {rosso, verde, giallo, arancio, marrone, nero}  colore;
typedef enum    {Giovanni, Claudia, Carla, Simone, Serafino}  persone;
typedef enum    {gen, feb, mar, apr, mag, giu, lug, ago, set, ott, nov, dic} mese;
```

```
persone individuo, individuo1, individuo2;

individuo = Giovanni;
if (individuo1 == individuo2) individuo = Claudia;
```

- Alcune osservazioni:
  - Spesso i valori del nuovo tipo sono rappresentati da nomi; però il compilatore associa a tali nomi nell'ordine un valore intero progressivo a partire da 0
  - Per esempio, per x di tipo mese:
    - gen è in realtà 0, apr è in realtà 3, ecc.

- Operazioni applicabili: le stesse degli interi

  Le seguenti operazioni:

  apr < giu

  rosso < arancio

  Producono come risultato un intero diverso da 0 (valore logico "vero"); mentre:

  dom < lun

  Simone < Giovanni

  Producono 0 (valore "false").

- Un importante caso particolare:

```
typedef   enum {false, true} boolean;
boolean    flag, ok;
```

- flag e ok possono così essere definite come variabili in grado di assumere valore vero (true) o falso (false) durante l'esecuzione di un programma che le usi.

- **NB: non invertire l'ordine!**

Tipo di dati strutturati

- Un tipo strutturato non è caratterizzato da un valore semplice, ma da informazione aggregata in diverse componenti.

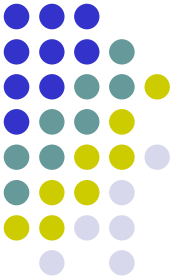
- Ad esempio un vettore o array consiste in una sequenza di elementi **consecutivi** e **omogenei**:

  - ```
    int matricole_studenti[24];     /*variabile di
                tipo array*/
    ```
  - ```
    Matricole_studenti[4] = 123444; /*assegnamento all'elemento
                    di indice 4*/
    ```

- Il linguaggio C non possiede tipi predefiniti strutturati, tuttavia ha 4 costruttori di tipo:
  - **array**
  - **struct**
  - **pointer**
  - **union**

- Consentono di definire tipi strutturati anche complessi

Il costruttore di tipo array (vettore)



- Definizione del nuovo tipo anArray:

```
typedef int anArray[20];
```

- Dichiazione di variabili di tipo anArray:

```
anArray lista1, lista2;
```

- La dichiarazione:

```
int lista[20];
```

- Va interpretata come *un'abbreviazione* per:

```
typedef int arrayAnonimo[20];
arrayAnonimo lista;
```

- Dichiarazione mediante nuovo tipo:

```
typedef double        VettoreDiReali[20];
VettoreDiReali        v1, v2, v3;
```

- Più semplice e altrettanto chiara:

```
double        v1[20], v2[20], v3[20];
```

- Dichiarazione mediante nuovi tipi:

```
typedef      double   PioggeMensili[12];
typedef      double   IndiciBorsa[12];
PioggeMensili      Piogge01, Piogge02, Piogge03;
IndiciBorsa    Indici01, Indici02, Indici03;
```

- Può essere preferibile a:

```
double   Piogge01[12], Piogge02[12], Piogge03[12],
         Indici01[12], Indici02[12], Indici03[12];
```

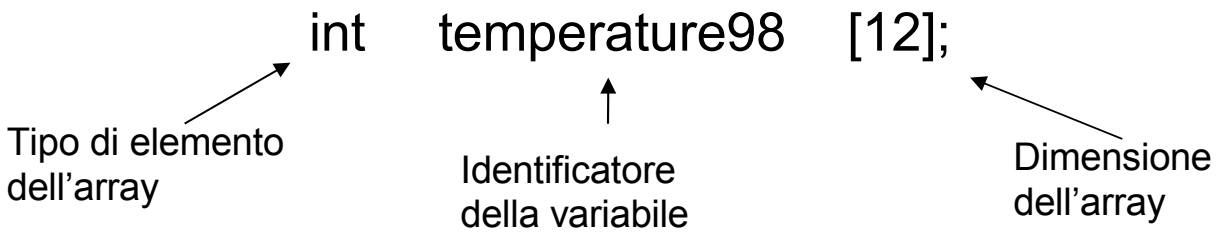
Quindi …

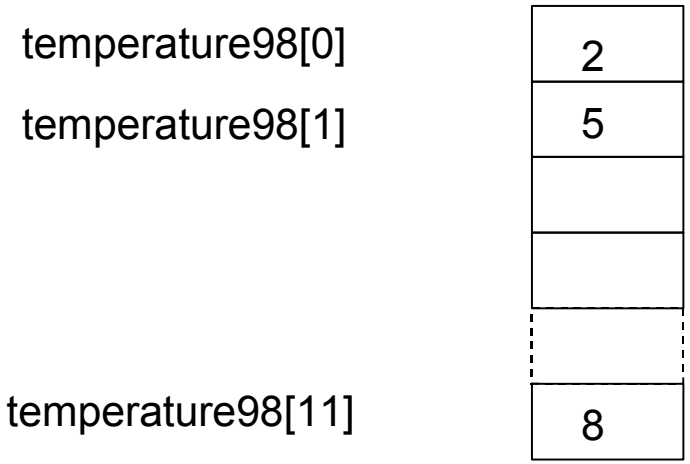
- Si può dichiarare esplicitamente un nuovo tipo mediante il costruttore array usando:
  - La parola chiave **typedef**
  - L'indicatore del tipo degli elementi dell'array: **semplice, strutturato , definito dall'utente**
  - Il nome del nuovo tipo ottenuto con il costruttore array
  - La dimensione dell'array tra parentesi quadre

- Se non esistono motivi per rendere <u>esplicita</u> la dichiarazione del tipo array la dichiarazione <u>implicita</u> viene sempre preferita
  - Evita spreco di tempo
  - E l'invenzione di nomi spesso inutili

- Gli elementi della variabile array:
  - Possono essere di un tipo semplice, strutturato, predefinito, definito dall'utente
  - Sono ordinati
  - Occupano celle di memoria contigue
  - Sono accessibili singolarmente tramite un indice che precisa la posizione dell'elemento voluto
  - Gli indici vanno da *0* ad *n-1*, dove *n* e' la dimensione
  - L'indice può essere una qualsiasi espressione di tipo int o char (non float!)

Esempio





- Indicizzati fra 0 ed n-1 se n è la dimensione dell'array
- Se si indicizza un elemento superiore ad n-1 non viene generato nessun errore
- La variabile usata per indicizzare può essere int, char, ma non un float

Matrici

- E' possibile definire un *array di array* (una *matrice*):

  typedef **int** Vettore[20];

  typedef Vettore MatriceIntera20Per20[20];

  MatriceIntera20Per20 matrice1;

- Oppure, più brevemente:

  typedef **int** MatriceIntera20Per20[20][20];
  MatriceIntera20Per20 matrice1;

- Anche una matrice può essere definita in modo *abbreviato*:

  int matrice1[20][20];

- E' possibile definire array di array di array…:

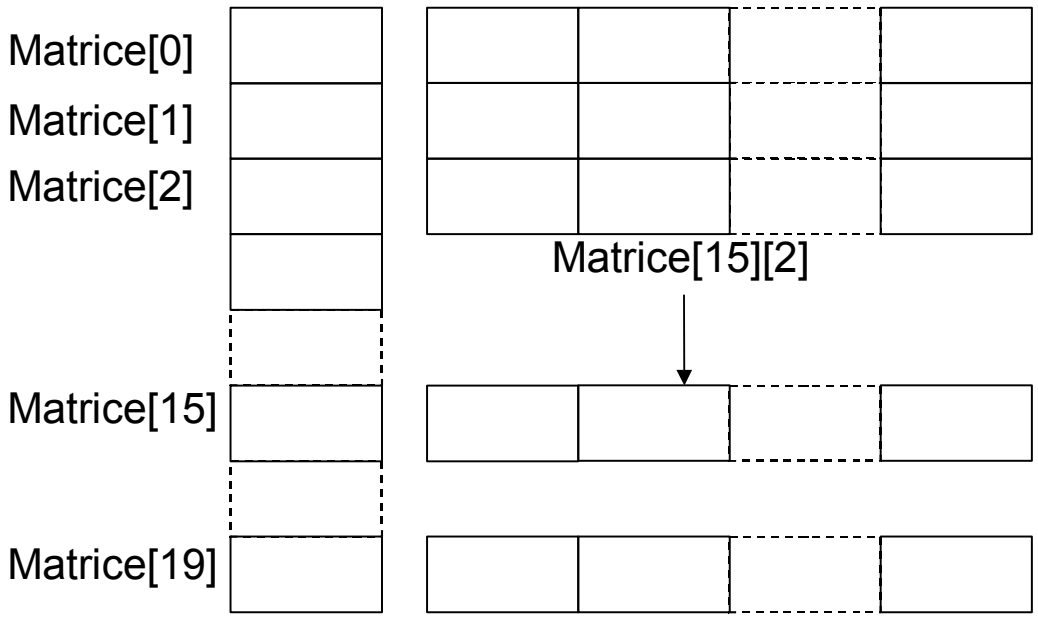
  int matriceTridimensionale1[10][20][30];

- Per accedere agli elementi di matriceTridimensionale1:

  matriceTridimensionale1[2][8][15]

- E' possibile definire array di …:

- Esempio
  - **int** matrice[20][20];



- L'elemento Matrice[i][j] corrisponde a quello posto sulla riga i nella colonna j
- Data la dichiarazione Matrice[n][m]
- L'indice delle righe varia fra 0 ed n-1
- L'indice della colonna varia fra 0 ed m-1

Esempio

```
int n = 9;
int m = 20;
int matrice[n][m];
int i;
int j;

main(){
   ...
   for (i=0; i<m; i++)
   {
      for (j=0; j<n; j++)
      {
         printf("%d", matrice[i][j]);
      }
      printf("\n");
   }
}
```

Dimensione degli array (1/3)

- Un array monodimensionale o multi-dimensionale ha sempre dimensioni fisse
  - int vettore[n]; /*errore*/
  - int vettore[10]; /*ok*/
- Molte volte la dimensione dell'array non è nota a priori
  - si pensi ad una serie di elementi inseriti da tastiera che terminano con zero
- In alcuni casi è possibile sovradimensionare, ma questo comunque comporta uno spreco di memoria
- Questi inconvenienti sono dovuti a questioni implementative: esecuzione più efficiente se la memoria è nota a priori

Dimensione degli array (2/3)

- Principio dell'*allocazione statica della memoria*
- Le celle astratte di memoria devono essere reperite nella memoria fisicamente disponibile della macchina
- Se questa operazione può essere fatta prima di cominciare, durante l'esecuzione del programma non si dovrà perdere tempo per cercare nuove celle fisiche da assegnare alle variabili strutturate che devono crescere in modo imprevisto
- I linguaggi di programmazione più usati confinano a situazioni eccezionali la possibilità di allocare nuovo spazione di memoria durante l'esecuzione del programma

Dimensione degli array (3/3)

- Esempio:
  - Stringhe (array di caratteri)

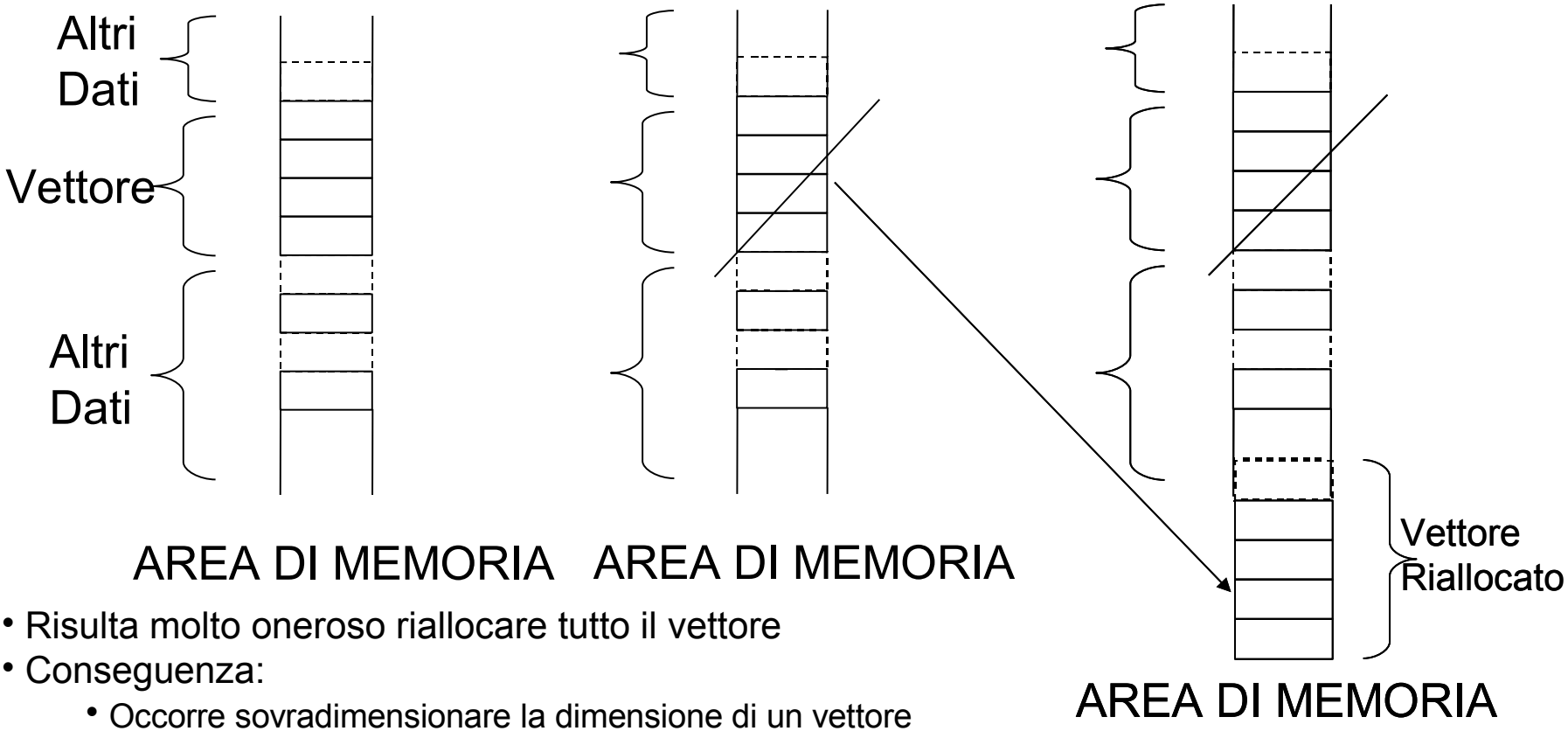
    typedef char string[30];

    string nome, cognome;

  - Nomi corti sprecherebbero spazio
  - Nomi lunghi non sarebbero rappresentabili

- **E' compito del programmatore assicurarsi che la sequenza di caratteri non ecceda la dimensione fissata**

Il costruttore di tipo array



- Risulta molto oneroso riallocare tutto il vettore
- Conseguenza:
  - Occorre sovradimensionare la dimensione di un vettore
  - Grande sovradimensionamento porta spreco di memoria
  - Se non si dimensiona bene c'è il rischio di overflow

Esempio

```
/* Programma per invertire una sequenza di 100 interi */
#include <stdio.h>
main()
{
        int contatore;
        int memorizzazione[100];

        contatore = 0;
        while (contatore < 100)
     {
                scanf("%d",&memorizzazione[contatore]);
                contatore = contatore + 1;
        }
        contatore = contatore -1;
        while (contatore >= 0)
     {
            printf("%d\n",memorizzazione[contatore]);
            contatore = contatore - 1;
        }
}
```

- Osservazioni:
  - Per una sequenza di 1000 interi la modifica del precedente programma è banale, ma allo stesso tempo pericolosa, poiché occorre sostituire tutti i 100 con 1000
  - E’ facile dimenticare qualche punto di sostituzione
  - Non tutti i 100 debbono in generale essere sostituiti quindi si potrebbero generare errori
  - La cosa migliore è allora definire una costante che contiene la lunghezza dell’array

```c
/* Programma per invertire una sequenza di 100 interi */

#include <stdio.h>
main()
{
        const int dim=100;

        int contatore;
        int memorizzazione[dim];

        contatore = 0;
        while (contatore < dim)
        {
            scanf("%d",&memorizzazione[contatore]);
            contatore = contatore + 1;
         }
        contatore = contatore -1;
        while (contatore >= 0)
        {
            printf("%d\n",memorizzazione[contatore]);
            contatore = contatore - 1;
         }
}
```

- Osservazioni
  - L’utilizzo di una costante fissa ugualmente la dimensione del vettore
  - E’ facile cambiare la dimensione del vettore
  - Naturalmente occorrerà ricompilare il programma

- Alternativamente alla costante può essere utilizzata la direttiva #define

```
#define  dim  100
```

- Contrariamente alla dichiarazione **const**, questa dichiarazione *non riserva spazio di memoria* e la sostituzione del valore costante al posto dell'identificatore viene eseguita dal prepocessore nel tempo di compilazione

```c
/* Programma per invertire una sequenza di 100 interi */

#include <stdio.h>
#define dim 100

main()
{
        int contatore;
        int memorizzazione[dim];

        contatore = 0;
        while (contatore < dim)
        {
            scanf("%d",&memorizzazione[contatore]);
            contatore = contatore + 1;
         }
        contatore = contatore -1;
        while (contatore >= 0)
        {
            printf("%d\n",memorizzazione[contatore]);
            contatore = contatore - 1;
         }
}
```

Infine …

- Gli elementi di un array possono essere coinvolti in tutte le operazioni proprie del tipo che li caratterizza e comparire in qualsiasi punto del programma in cui può comparire una variabile o espressione dello stesso tipo

```
int v[10];
int v1[10];
……….
if ( v[4] > v1[5] ) ……….
```

- Ma l' identificatore globale non può essere coinvolto in assegnamenti o confronti
  - v1=v; /*scorretto*/

- Per assegnare gli elementi di un array occorre trattarli singolarmente

```
int Array1[10];
int Array2[10];
```

risulta scorretto il tentativo di assegnare in blocco tutti i valori dei singoli elementi dell'array Array1 con gli elementi dell'Array2 tramite l'istruzione

```
Array2=Array1;
```

Il modo giusto di operare è il seguente:

```
for (i=0;i<10;i++){
    Array2[i]=Array1[i];
}
```

Stringhe (1/2)

- Non esiste un tipo predefinito stringa in C
- Le stringhe sono rappresentate come un array di caratteri terminante con il carattere speciale '\0'

Alla dichiarazione è possibile assegnare un valore alle variabili:

- **int** k = 0;
- **char** nome [ ] = "emanuele";
- **char** nome[9];
- "x" è diverso da 'x'

char nome[9];

……

nome[0] = 'e' ;
nome[1] = 'm' ;
nome[2] = 'a' ;
nome[3] = 'n' ;
nome[4] = 'u' ;
nome[5] = 'e' ;
nome[6] = 'l' ;
nome[7] = 'e' ;
nome[8] = '\0' ;

……

nome[7]='a';

e
m
a
n
u
e
l
e
\0

Stringhe (2/2)

- **NB: Una stringa è un array di caratteri, ma un array di caratteri non è necessariamente una stringa: per esserlo occorre che l'ultimo elemento sia il carattere '\0'**

- **char** ac[20];

  L'esempio mostra la dichiarazione di un array di caratteri, senza specificare il suo contenuto. Per il momento non si può parlare di stringa, soprattutto perchè per essere tale, la stringa deve contenere dei caratteri.

- **char** ac[] = { 'c', 'i', 'a', 'o' };

  Questo esempio mostra la dichiarazione di un array di quattro caratteri. All'interno delle parentesi quadre non è stata specificata la dimensione perchè questa si determina dall'inizializzazione. Anche in questo caso non si può ancora parlare di stringa, perchè manca la terminazione.

- **char** acz[] = { 'c', 'i', 'a', 'o', '\0' };

  Questo esempio mostra la dichiarazione di un array di cinque caratteri corrispondente a una stringa vera e propria. L'esempio seguente è equivalente, solo che utilizza una rappresentazione più semplice.

- **char** acz[] = "ciao";

Esempio

- Programma che chiede in input 10 interi, li memorizza in un array e poi stampa il valore della somma degli elementi

```c
#include <stdio.h>
#define SIZE 10

main(){
   int array[SIZE];
   int i = 0, somma = 0;
   for( i = 0; i < SIZE; i++ ){
    printf("inserire l'intero %d: ", i+1);
    scanf("%d", &array[i]);
   }
   for( i = 0; i < SIZE; i++ )
    somma = somma+array[i];    //somma+=array[i];
   printf("La somma vale: %d\n", somma);

}
```

Array palindromo

- Scrivere un programma che chiede in input n interi, li memorizza in un vettore e successivamente verifica se l'array è palindromo
- Si ricorda che una parola palindroma è una parola che si può leggere indifferentemente da sinistra a destra e viceversa, es: osso, alla
- Allo stesso modo, nel caso di array:

-1	4	6	7	6	4	-1

```c
#include <stdio.h>
#define DIMARRAY 6

int array[DIMARRAY], n = 0;

main(){
   for ( n=0; n < DIMARRAY; n++ ) {
    printf("Inserire il valore %d ", n+1 );
    scanf("%d", &array[n] );
   }
   for (n=0; n<DIMARRAY/2&&(array[n]==array[DIMARRAY-n-1]);n++ );
   if ( n == DIMARRAY/2 )
    printf("Il vettore inserito e' palindromo\n");
   else
    printf("Il vettore inserito non e' palindromo\n");
}
```