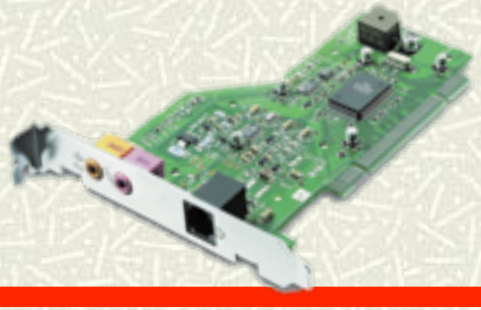


Informatica Generale

Linguaggio C

Array Semplici e Direttive per il Compilatore

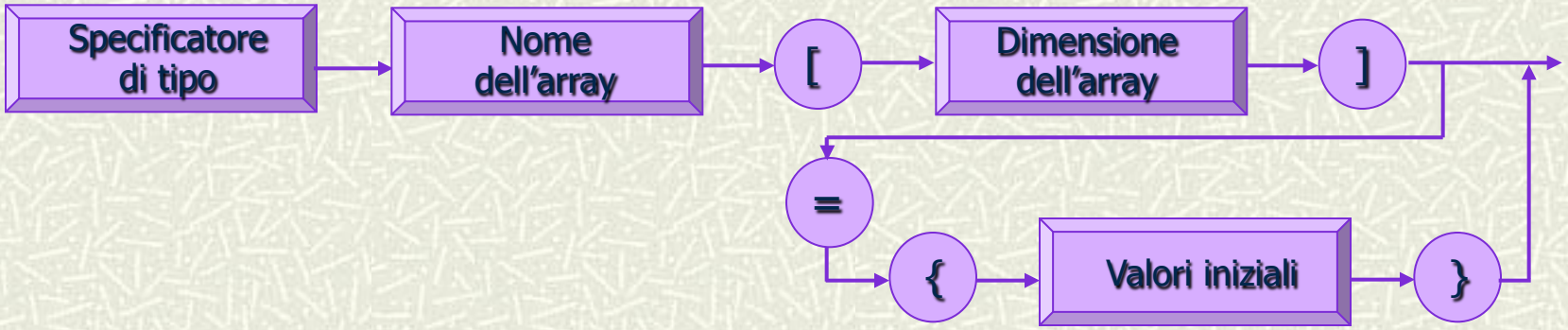
- Array
  - Mondimensionali
  - Multidimesionali (Matrici)
- Preprocessore
- Commenti

Gli array

- Nel linguaggio C, un **array** è un insieme di variabili dello stesso tipo memorizzate consecutivamente
- Ogni variabile dell'array, detta **elemento**, può essere acceduta mediante il nome dell'array ed un **indice** (un espressione), con il primo elemento individuato dall'indice di valore 0
- **Esempio**: Memorizzare la temperatura media registrata per ogni giorno dell'anno
  - Si dichiara un array di 365 elementi (piuttosto che 365 variabili!), identificati con un solo nome, e che occupano posizioni di memoria consecutive
- **Nota**: Gli array contengono informazione correlata (le temperature di un anno, i voti di esame di uno studente, etc.)

La dichiarazione di array - 1

- La sintassi della **dichiarazione di un array** è:



- La dichiarazione viene effettuata inserendo una coppia di parentesi quadre dopo il nome dell'array
- La dimensione dell'array viene specificata inserendo il numero di elementi all'interno delle parentesi quadre

La dichiarazione di array - 2

- Il **riferimento ad un elemento dell'array** è invece l'operazione di accesso al dato elemento ed avviene utilizzando il nome dell'array seguito dall'indice dell'elemento (racchiuso fra parentesi quadre)
- Le istruzioni di dichiarazione di un array e di riferimento ad un elemento dell'array sono molto simili nella forma, ma molto diverse nel significato

```
/* La seguente è una dichiarazione;
  * il valore 365  specifica il numero
  * di elementi dell'array
  */
int daily_temp[365];
```

```
/* I seguenti sono riferimenti a elementi
  * dell'array; i valori 0,1,2,… specificano
  * gli elementi a cui accedere
  */
daily_temp[0] = 2;
daily_temp[1] = 5;
daily_temp[2] = 3;
… … …
```

La dichiarazione di array - 3

- Esempio: Calcolo della temperatura media annua

```
#include <stdio.h>
#include <stdlib.h>
#define DAYS_IN_YEAR 365

main()
{
        int j, sum=0;
        int daily_temp[DAYS_IN_YEAR];

        /* si assegnano i valori a daily_temp[] */

        for (j=0; j<DAYS_IN_YEAR; j++)
            sum += daily_temp[j];
        printf("La temperatura media dell'anno è %d.\n", sum/DAYS_IN_YEAR);
        exit(0);
}
```

La modalità di memorizzazione degli array

- **Esempio**

```
int ar[5];   /* dichiarazione */
ar[0] = 15;
ar[1] = 17;
ar[3] = 3;
```

- ar[2] e ar[4] sono indefiniti: il contenuto delle posizioni di memoria è quello rimasto da esecuzioni precedenti (**garbage**)

L'inizializzazione di array - 1

- La presenza di valori indefiniti in alcuni elementi dell'array può provocare errori difficili da rilevare

  ⇒ Occorre inizializzare l'intero vettore

  - Dichiarare l'array **static** (vettore **a durata fissa**): gli elementi del vettore, non inizializzati esplicitamente, vengono posti a zero
  - Valori diversi possono essere specificati, facendoli seguire alla dichiarazione dell'array, racchiusi fra parentesi graffe: tali valori devono essere espressioni costanti che possano essere convertite automaticamente nel tipo dell'array

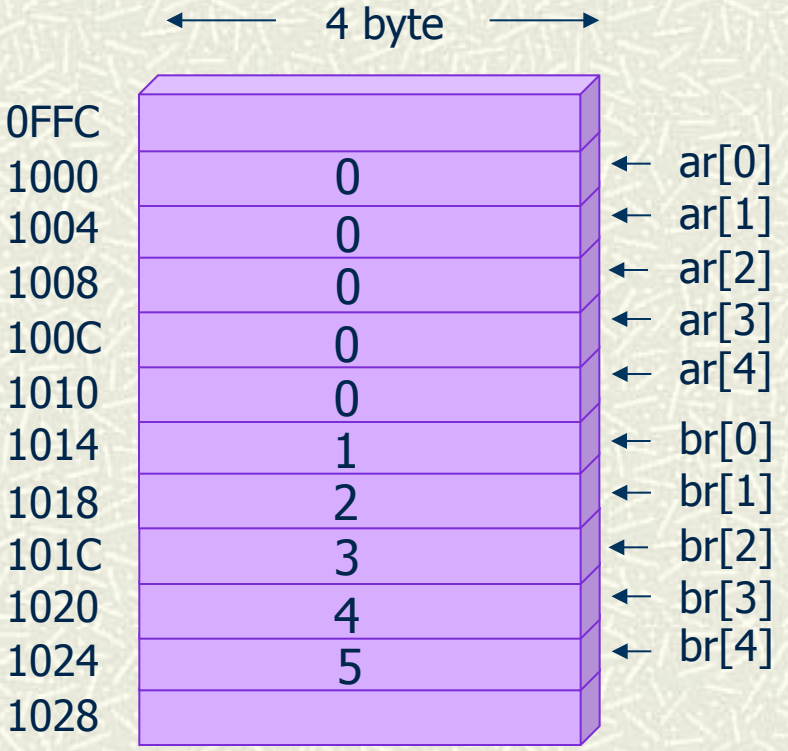
L'inizializzazione di array - 2

- **Esempio**

  static int ar[5];

  static int br[5] = {1,2,3.5,4,5};

- **Nota**: il valore floating-point 3.5 è convertito nel valore intero 3



L'inizializzazione di array - 3

- Specificare un numero maggiore di valori di inizializzazione, rispetto agli elementi dell'array, costituisce un errore segnalato dal compilatore
- Se vengono specificati meno valori rispetto alla dimensione, gli elementi rimanenti vengono inizializzati a zero
- Esempio: La dichiarazione

```
static int cr[5] = {1,2,3};
```

produce l'inizializzazione

```
cr[0] = 1
cr[1] = 2
cr[2] = 3
cr[3] = 0
cr[4] = 0
```

L'inizializzazione di array - 4

- Se vengono specificati i valori iniziali, può essere omessa la dimensione dell'array: il compilatore calcola automaticamente il numero degli elementi sulla base del numero dei valori iniziali specificati
- **Esempio**:

```
static char dr[] = {'a', 'b', 'c', 'd'};
```

comporta la creazione di un array di quattro elementi, di tipo **char**, caratterizzati dai valori iniziali

```
dr[0] = 'a'
dr[1] = 'b'
dr[2] = 'c'
dr[3] = 'd'
```

Esempio 1:

- Copia il vettore vett_iniz nel vettore vett_fin

```
#include <stdio.h>
#define NUMDATI 5

int  vett_iniz[NUMDATI] = {11, -2, -63, 4, 15};
int  vett_fin[NUMDATI], indice;

main(){
    for (indice = 0; indice < NUMDATI; indice++)
            vett_fin[indice] = vett_iniz[indice];
}
```

Esempio 2:

- Leggere 10 valori da tastiera e memorizzarli in un vettore; quindi calcolarne il minimo ed il massimo.

Pseudocodice:

- *Con un indice* ind *che varia tra 0 e 9: legge un dato e lo salva in* vettdati[ind]*;*
- *Inizializzo la variabile* massimo *e la variabile* minimo *col primo elemento del vettore* vettdati[0]*;*
- *Con un indice* ind *che varia tra 1 e 9:*
  se vettdati[ind] *è più grande di* massimo*:*
  massimo = vettdati[ind]*;*
  altrimenti se vettdati[ind] *è più piccolo di* minimo*:*
  minimo = vettdati[ind]*;*
- *Visualizza* massimo *e* minimo

Codice c

```
#include <stdio.h>
#define NUMDATI 10
main(){
int  minimo, massimo, ind;
int vettdati[NUMDATI];
/*        lettura dei dati        */
for (ind = 0; ind < NUMDATI; ind++)
   {
   printf (“\nIntroduci vettdati[%d]: ", ind);
   scanf ("%d", &vettdati[ind]);
   }
```

Seconda parte

```
/*                    cerca il massimo e il minimo                    */
massimo = vettdati[0];
minimo = vettdati[0];
for (ind = 1; ind < NUMDATI; ind++) {
      if  (vettdati[ind] > massimo)
                  massimo = vettdati[ind];
      else {
                  if (vettdati[ind] < minimo)
                              minimo = vettdati[ind];
                }
   }
printf  ("\nIl massimo è %d e il minimo è %d\n ", massimo, minimo);
}
```

Uscita dal limite superiore di un array

- Il compilatore, di solito, non controlla che l'accesso agli elementi di un array venga effettuato rispettandone i limiti dimensionali

  ⇨ È possibile accedere per errore ad elementi per i quali non è stata allocata memoria (aree di memoria riservate ad altre variabili, riservate ad altri processi, etc.)

- **Esempio**:

```
main()
{
        int ar[10], j;

        for (j=0; j<=10; j++)
            ar[j] = 0;
}
```

Essendo ar un array di 10 elementi, quelli cui è possibile accedere in modo corretto hanno indice da 0 a 9: il ciclo **for** contiene un errore *off-by-one*

Probabilmente verrebbe azzerata la variabile j ⇨ il ciclo diventa infinito


Le stringhe

Definizione

- Una **stringa** è un array di caratteri terminato dal carattere nullo, corrispondente alla sequenza di escape **\0** (con valore numerico associato zero)
- Una **stringa costante** (o **letterale**) è una serie di caratteri racchiusi fra doppi apici: tale stringa è di tipo **array di caratteri**, con ogni carattere che occupa un byte
- Ad ogni stringa viene aggiunto automaticamente dal compilatore un carattere nullo, ad indicarne la fine

Dichiarazione e inizializzazione - 1

- Per memorizzare una stringa occorre dichiarare un array di **char**, che può essere inizializzato con una stringa costante

  static char str[]="testo";

- L'array ha dimensione maggiore di uno rispetto alla lunghezza della stringa, per consentire la memorizzazione del carattere nullo di terminazione (**str** ha lunghezza 6 byte)
- Il compilatore segnala un errore se si dichiara la lunghezza della stringa n, e si inizializza con una stringa costante di lunghezza >n

  static char str[3]="quattro"; /* SCORRETTO */

  static char str1[3]="tre"; /* NON CORRETTO */

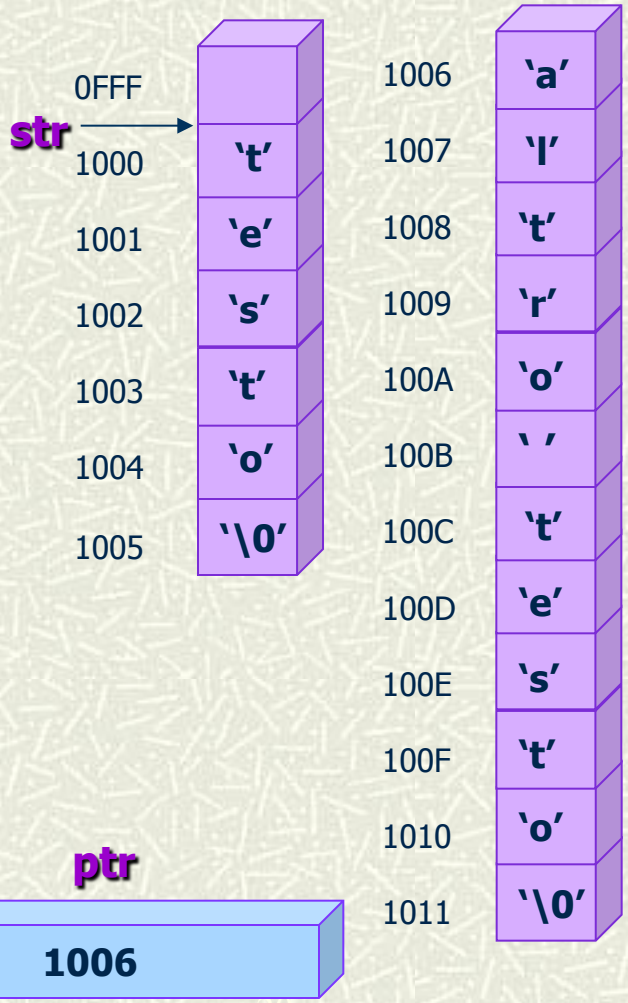
NB: non da errore ma lo '/0' non viene incluso con tutti i problemi derivanti nell'aver un array di caratteri (e non una stringa).

Dichiarazione e inizializzazione - 2

- È possibile inizializzare un puntatore a **char** con una stringa costante:

  char *ptr = "altro testo";

  si crea un array di caratteri, inizializzato ad "altro testo", riservando però memoria anche per il puntatore
  - Nel caso dell'array, tutti i successivi accessi utilizzano il nome dell'array come riferimento per l'indirizzo dell'elemento iniziale dell'array: tale indirizzo non può essere modificato
  - Il puntatore è una variabile e può essere modificato: l'indirizzo relativo alla prima inizializzazione viene perso



Lettura e scrittura di stringhe

- **Esempio**: Scrivere un programma che legge una stringa dalla periferica d'ingresso di default e la stampa dieci volte

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_CHAR 80

main()
{
        char str[MAX_CHAR];
        int i;

        printf("Introdurre una stringa:");
        scanf("%s", str);
        for (i=0; i<10; i++)
            printf("%s\n", str);
        exit(0);
}
```

È possibile utilizzare il nome dell'array come argomento per le funzioni di I/O, in quanto puntatore all'inizio dell'array

Gli array multidimesionali - 1

- Un array di array è un **array multidimensionale** e viene dichiarato per mezzo di una sequenza di coppie di parentesi quadre

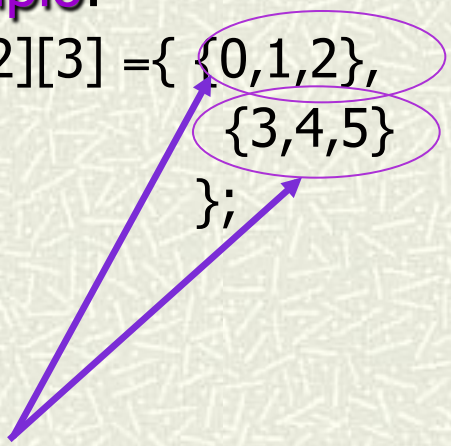
```
/* x è un array di tre elementi costituiti
 * da array di cinque elementi
 */
int x[3][5];
```

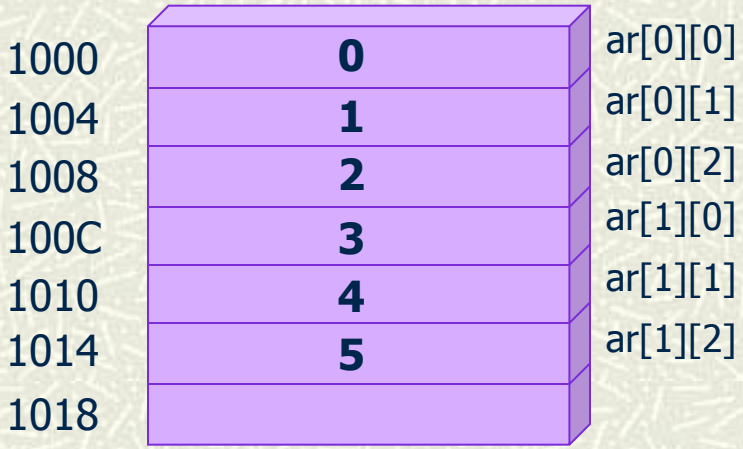
- Anche se un array multidimensionale viene memorizzato come una sequenza di elementi, può essere manipolato come un array di array
- Per accedere ad un elemento di un array multidimensionale occorre specificare tanti indici quante sono le dimensioni dell'array

Gli array multidimensionali - 2

- Gli array multidimensionali sono memorizzati con precedenza delle righe, cioè l'ultimo indice varia più velocemente
- **Esempio**:

```
int a[2][3] ={ {0,1,2},
               {3,4,5}
             };
```





Nell'inizializzazione, ogni riga di valori è racchiusa fra parentesi graffe (in questo caso, servono per migliorare la leggibilità)

Vettori multidimensionali

Esempio:

matrice bidimensionale di numeri interi formata da tre righe e 5 colonne:

```
int a[3][5];
```

a

a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]	a[0]
a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]	a[1]
A[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]	a[2]

Vettori multidimensionali

- Accesso ad un elemento:

  <nome vettore> [*<posizione1>*] [*<posizione2>*].....

- Per esempio: **matrix [10][20][15]**

  individua l'elemento di coordinate rispettivamente 10, 20 e 15 nella matrice a 3 dimensioni matrix.

- Inizializzazione di un vettore multidimensionale:
  - deve essere effettuata per righe!

```
int a[3][2]  =  {       {8,1},          /*   vett[0]   */
                        {1,9},          /*   vett[1]   */
                        {0,3}           /*   vett[2]   */
                };
```

Vettori multidimensionali e cicli

- Per un vettore a più dimensioni, la scansione va applicata a tutte le dimensioni: in questo caso si devono in genere utilizzare "*cicli annidati*".

- Esempio: elaborazione degli elementi di un vettore bidimensionale.

```
int vett [3][5];
...
for (i = 0; i < 3; i++)  {                  /*     per ogni riga     */
        for (j = 0; j < 5; j++) {           /*   per ogni colonna    */
          ...  elaborazione su  vett[i][j]
        }
}
```

I commenti - 1

- Un commento è un testo che viene incluso in un file sorgente per spiegare il significato del codice
- I commenti sono ignorati dal compilatore
- I commenti sono un elemento fondamentale nello sviluppo del software: il linguaggio C consente di inserire commenti racchiudendo il testo tra i simboli /* e */ (oppure // per commentare un'intera riga)

```
/* Questa funzione restituisce
*  il quadrato del suo argomento
*/
int square(num)
int num;
{
   int answer;

   answer=num*num;  /* non si controlla l'overflow */
   return answer;
}
```

I commenti - 2

- Non sono ammessi commenti innestati
- Un commento può occupare più linee
- Affinché i commenti non interrompano il flusso di un programma…
  - …occorre dedicare ai commenti intere linee di codice
  - …o collocarli sulla destra del codice, quando condensabili in un'unica riga
- Cosa deve essere commentato? Tutto ciò che non è ovvio:
  - Espressioni complesse, strutture dati e scopo delle funzioni
  - Eventuali modifiche apportate al programma, per poterne tenere traccia
- In particolare, ogni funzione dovrebbe avere un commento di intestazione, che descrive "cosa fa" la funzione ed il significato dei suoi parametri

I commenti - 3

- Tuttavia, commenti con scarso contenuto informativo possono rendere un programma difficile da leggere
- Un esempio di stile di documentazione scadente…

```
j=j+1;    /* incrementa j */
```

- Inoltre, commenti molto lunghi non compensano codice illeggibile o stilisticamente imperfetto

Il preprocessore

- Il preprocessore C è un programma che viene eseguito prima del compilatore (non è necessario "lanciarlo" esplicitamente)
- Attraverso il preprocessore si esprimono direttive al compilatore
- Il preprocessore ha la sua grammatica e la sua sintassi che sono scorrelate da quelle del C
- Ogni direttiva inizia con il simbolo **#**, che deve essere il primo carattere diverso dallo spazio sulla linea
- Le direttive del preprocessore terminano con un newline (non con ";")

Direttive del preprocessore

- Principali compiti richiesti al preprocessore:
  - Inclusione del codice sorgente scritto su altro file
  - Definizione delle costanti simboliche
  - Compilazione condizionale del codice

La direttiva #include - 1

- La direttiva **#include** fa sì che il compilatore legga il testo sorgente da un file diverso da quello che sta compilando
- **#include** lascia inalterato il file da cui vengono prelevati i contenuti
  - Utile quando le stesse informazioni devono essere condivise da più file sorgente: si raccolgono le informazioni comuni in un unico file e lo si include ovunque sia necessario
  - Si riduce la quantità di testo da digitare e si facilita la manutenzione: i cambiamenti al codice condiviso hanno effetto immediato su tutti i programmi che lo includono

La direttiva #include - 2

- La direttiva **#include** può assumere due formati

#include <nome_file.h>

#include "nome_file.h"

  - ...nel primo caso, il preprocessore cerca il file in una directory speciale, definita dall'implementazione del compilatore, dove sono contenuti i file che vengono normalmente inclusi da tutti i programmi utente (sintassi usata per includere file di intestazione, ***header file***, della libreria standard)
  - ...nel secondo caso, il file viene prima cercato nella directory del file sorgente e, quando non reperito, seguendo il percorso classico

La direttiva #define

- La direttiva #define consente di associare un nome ad una costante
- Esempio:

```
#define  NIENTE  0
```

  associa il nome "NIENTE" alla costante 0
- Per evitare confusione fra nomi di costanti e nomi di variabili, è pratica comune usare solo lettere maiuscole per le costanti e solo minuscole per le variabili
- L'associazione di nomi alle costanti permette…
  - …di utilizzare un nome descrittivo per oggetti altrimenti non autoreferenziali
  - …di semplificare la modifica del software: cambiare il valore ad una costante equivale a cambiarne la sola definizione e non tutte le occorrenze