ELEMENTI DI LINGUAGGIO C

Fondamenti di Programmazione 2022/2023

Francesco Tortorella

Il linguaggio C

Early C

- 1969: B created, based on BCPL, to replace PDP-7 assembler as the system programming language for Unix
- 1971: NB ("new B") created when porting B to PDP-11
- 1972: Language renamed to C
- 1973: Unix re-written in C
- 1978: The C Programming Language, 1st edition

Standard C

- 1983: ANSI established X3J11 committee
- 1988: The C Programming Language, 2nd edition
- 1989: C89, the ANSI C standard published
- 1990: C90, the ANSI C standard accepted as ISO/IEC 9899:1990
- 1995: C95 (ISO/IEC 9899:1990/Amd.1:1995) (online store)
- 1999: C99 (ISO/IEC 9899:1999)

Il linguaggio C

```
/* Semplice programma in C */
#include <stdio.h>


int main()
{
   printf("Salve, mondo!\n");
   return (0);
}
```

Sistema dei tipi…

- Il linguaggio C prevede che siano esplicitamente dichiarato i tipi di tutte le variabili.
- In C sono disponibili vari tipi di dato.
- **Tipi numerici**:
  - `int`
  - `float`
  - `double`
- **Tipi non numerici**:
  - `char`
  - `bool`
  - `void`

Il tipo `int`

- È costituito da un sottoinsieme limitato dei numeri interi
- Caratteristiche:
  - Dimensione: 32 bit (Nota bene: la dimensione dipende dal compilatore, in questo caso `gcc 10.2.0`)
  - Valore minimo: -2147483648
  - Valore massimo: +2147483647
- Operazioni ammesse:
  - Assegnazione =
  - Somma +
  - Sottrazione -
  - Moltiplicazione *
  - Divisione /
  - Modulo %
  - Confronto > < >= <= == !=

Il tipo `float`

- È costituito da un sottoinsieme limitato e discreto dei numeri reali
- Caratteristiche:
  - Dimensione: 4 bytes (Nota bene: la dimensione dipende dal compilatore, in questo caso `gcc 10.2.0`)
  - Valore minimo (abs): 3.4E- 38
  - Valore massimo (abs): 3.4E+38
- Operazioni ammesse:
  - Assegnazione =
  - Somma +
  - Sottrazione -
  - Moltiplicazione *
  - Divisione /
  - Confronto > < >= <= == !=

Il tipo `double`

- È costituito da un sottoinsieme limitato e discreto dei numeri reali, ma con range e precisione maggiore rispetto a float (doppia precisione)
- Caratteristiche:
  - Dimensione: 8 bytes (Nota bene: la dimensione dipende dal compilatore, in questo caso `gcc 10.2.0`)
  - Valore minimo (abs): 1.7E- 308
  - Valore massimo (abs): 1.7E+308
- Operazioni ammesse:
  - Assegnazione =
  - Somma +
  - Sottrazione -
  - Moltiplicazione *
  - Divisione /
  - Confronto > < >= <= == !=

Il tipo `char`

- Consiste in un insieme di caratteri, alcuni stampabili (caratteri alfabetici, cifre, caratteri di punteggiatura, ecc.) ed altri non stampabili tramite i quali si gestisce il formato dell'input/output (caratteri di controllo).
- I sottoinsiemi delle lettere e delle cifre sono ordinati e coerenti.
- Per la rappresentazione interna, viene tipicamente usato il codice ASCII, che mette in corrispondenza ogni carattere con un numero intero compreso tra 0 e 255.

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Il tipo `char`

- Di fatto anche **`char`** è un tipo numerico
- Caratteristiche:
  - Dimensione: 1 byte (Nota bene: la dimensione dipende dal compilatore, in questo caso **`gcc 10.2.0`**)
  - Valore minimo: -128
  - Valore massimo: +127
- Sono permesse le operazioni aritmetiche tipiche degli interi

Il tipo `bool`

- È un tipo costituito dai due soli valori **`false`** e **`true`**, corrispondenti a falso e vero e rappresentati da 0 e 1. Il tipo rappresenta le informazioni di tipo logico (es. il risultato di un confronto, il verificarsi di una situazione).
- Caratteristiche:
  - Dimensione: 1 byte
  - Valore minimo: **`false`**
  - Valore massimo: **`true`**
- Operazioni ammesse
  - assegnazione =
  - disgiunzione ||
  - congiunzione &&
  - negazione !

Per utilizzare il tipo **`bool`**, è necessario inserire la direttiva **`#include <stdbool.h>`**

Modificatori di tipo

- Sono usati per creare nuovi tipi modificando i tipi base.
- **`signed`** e **`unsigned`**: senza ulteriori specificazioni (default), i tipi sono signed. Con il modificatore **`unsigned`**, il tipo è in grado di contenere soltanto valori non negativi
- **`unsigned char`**: 0…255
- **`unsigned int`**: 0…4294967295

Modificatori di tipo

- **`short`** e **`long`**: modificano l'estensione del tipo
  - **`short int`**: 2 byte
  - **`long int`**: 4 byte
  - **`long double`**: 16 byte
  - Nota bene: la dimensione dipende dal compilatore, in questo caso **`gcc 10.2.0`**
- I modificatori possono combinarsi:
  - **`unsigned short int`**
  - **`unsigned long int`**

Tipo enumerativo

- La sintassi per dichiarare il tipo enumerativo è

```
enum Colore { RED, GREEN, BLUE };
```

- Il nome del tipo da utilizzare dalla dichiarazione del tipo in poi è:

```
enum Colore
```

Variabili

- Per usare una variabile, questa deve essere dapprima **definita**.
- La definizione rende disponibile (da qualche parte in memoria) la variabile che mantiene il tipo assegnato nella definizione fino al termine del programma.
- La sintassi è *<tipo> nome_variabile*;
- Esempi:
  - `int a;`
  - `int a,b,c;`
  - `float x,y,z;`
  - `enum Colore col;`

Definizione di variabili

- Il nome della variabile non può coincidere con una delle parole chiave riservate del C:

```
auto double int struct break else long switch
case enum register typedef char extern return
union const float short  unsigned continue for
signed void default goto sizeof volatile do if
static while _Bool _Imaginary restrict _Complex
inline
```

Definizione di variabili

- I caratteri ammessi sono lettere, cifre e carattere di sottolineatura (underscore _), messi in qualunque ordine, purché il primo carattere del nome sia una lettera o l'underscore (sconsigliato).
- C'è differenza tra caratteri minuscoli e maiuscoli (case sensitive), per cui **`a`** e **`A`** sono due variabili diverse.
- Nello scegliere il nome per le variabili, è consigliabile orientarsi verso nomi significativi del ruolo della variabile nel programma.

Esempi

- Definizioni corrette:
  - `int Pippo, a31;`
  - `float  pippo, radice_equazione;`
  - `double Vercingetorige;`
  - `enum Colore x;`
- Definizioni errate:
  - `double 27pluto;`
  - `int conta num;`
  - `Float x,y,z;`

Costanti

- Il C prevede tre modalità per definire delle costanti:
  - **Costanti letterali** (literals)
  - **Costanti definite** (**`#define`**)
  - **Costanti dichiarate** (**`const`**)

Costanti letterali

- **Il valore della costante è rappresentato direttamente**.
- Costanti di tipo **`int`**
  - Sono definite come sequenze di cifre decimali, eventualmente precedute da un segno (+ o -):
    0 -1 3256 +34 12L 33U 5321UL 0713 0X12FF 0XFUL
- Costanti di tipo **`float`**
  - Sono definite come sequenze di cifre decimali, eventualmente precedute da un segno (+ o -), strutturate in virgola fissa o in virgola mobile (floating point):
    0.1 -3.7 0.0001 1.0E-4 -7.6E12 4.

Costanti letterali

- Costanti di tipo **`char`**
  - sono definite come caratteri racchiusi tra singoli apici ('):
    'x' 'A' '\n' '\t' '2'
- Costanti di tipo stringa di caratteri
  - sono definite come sequenze di caratteri racchiusi tra doppi apici ("):
    "Pippo" "Valore di x: " "x"
- Costanti di tipo **`bool`**
  - sono solo due: **`false`** e **`true`**

Costanti definite

- Viene utilizzata la direttiva **`#define`** che permette di associare **testualmente** un valore ad un identificatore

```
#define MAX   12
#define PI    3.14
```

- All'atto della compilazione il **preprocessore** sostituisce ogni occorrenza dell'identificatore con il valore corrispondente

Costanti definite

```
#define MAX 10

int main(){
   int a,b,i;

   a = MAX;
   b = MAX/2 + 1;
   i = 0;

   while(i < MAX)
       b = b+1;
  }
```


```
int main(){
   int a,b,i;

   a = 10;
   b = 10/2 + 1;
   i = 0;

   while(i < 10)
       b = b+1;
  }
```

Costanti dichiarate

- Il valore viene associato ad un identificatore e ne viene specificato anche il tipo:

```
const int MIN=0;
const float PI=3.14;
```

- In questo caso l'identificatore è a tutti gli effetti una variabile non modificabile.

- Qual è la differenza?

```
const int pippo=1;

#define pippo 1
```

Sistema degli operatori (sottinsieme)

Precedenza	Operatore	Descrizione	Associatività
1	()	Chiamata di funzione	Da sinistra a destra
	[]	Accesso ad array	
2	+ -	+ e - unari	Da destra a sinistra
	!	NOT logico	
	(*tipo*)	Cast	
	*		
	&		
3	* / %	Moltiplicazione, divisione, resto	Da sinistra a destra
4	+ -	Addizione e sottrazione	
5	< <=	Operatori relazionali < e ≤	
	> >=	Operatori relazionali > e ≥	
6	== !=	Operatori relazionali = e ≠	
7	&&	AND logico	
8	||	OR logico	
9	=	Assegnazione	Da destra a sinistra

Espressioni

- Un’espressione consiste in un operando o in una combinazione di operandi e operatori.
- La valutazione di un’espressione porta al calcolo di un valore appartenente ad un tipo specifico.
- Esempi:

```
2+3    2.1*3.5+pigreco    a>=2

(7+2)/3    (a>2)&&(b==0)   !trovato
```

Espressioni

- Nel caso ci siano operandi di tipo diverso, l'espressione assume il tipo "più ampio" tra quelli presenti.
- Esempio:
  `int` *op* **`float`** → **`float`**
  7+5*2.4 → 7+12.0 → 19.0
- **Achtung!** Qual è il valore dell'espressione?
  7.5+1/2
- Questo effetto va sotto il nome di **casting implicito**.

Espressioni

- **Alcuni casi di casting implicito**

  `int` *op* `float` → `float`

  `char` *op* `int` → `int`

  `float` *op* `double` → `double`

  `int` *op* `double` → `double`

Espressioni

- È possibile modificare il **valore del tipo** di un'espressione, indicando esplicitamente il tipo da impiegare.
- Esempio:
  `7.5+(float)1/2` → `7.5+1.0/2`
- Questa operazione va sotto il nome di **casting esplicito**.

Istruzioni di calcolo e assegnazione

- L'effetto è di aggiornare il valore di una variabile di un certo tipo con il valore ottenuto dalla valutazione di un'espressione dello stesso tipo.
- Il formato è:

```
variabile = espressione;
```

left_value ← right_value

- Esempi:

```
a=4;   a=a+1;  cond = x > y;
b=0;   a=a+b;  cond =(a>=0)&&(a<=9);
b=a;
```

Istruzioni di calcolo e assegnazione

- Quali sono istruzioni corrette?

```
int i, j, val_m;
const int ci = i;

2040 = val_m;
i + j = val_m;
ci = val_m;
i = j;
```

Autoincremento e autodecremento

istruzione	istruzione equivalente	restituisce	
++x;	x=x+1;	variabile	preincremento
x++;	x=x+1;	valore	postincremento
--x;	x=x-1;	variabile	predecremento
x--;	x=x-1;	valore	postdecremento
y=++x;	x=x+1;y=x;		
y=x++;	y=x;x=x+1;		

Operazioni di Input/Output

- Con le operazioni di **input**, il valore di una variabile viene modificato con il valore ottenuto grazie ad un'operazione di lettura dall'unità di ingresso (tastiera).

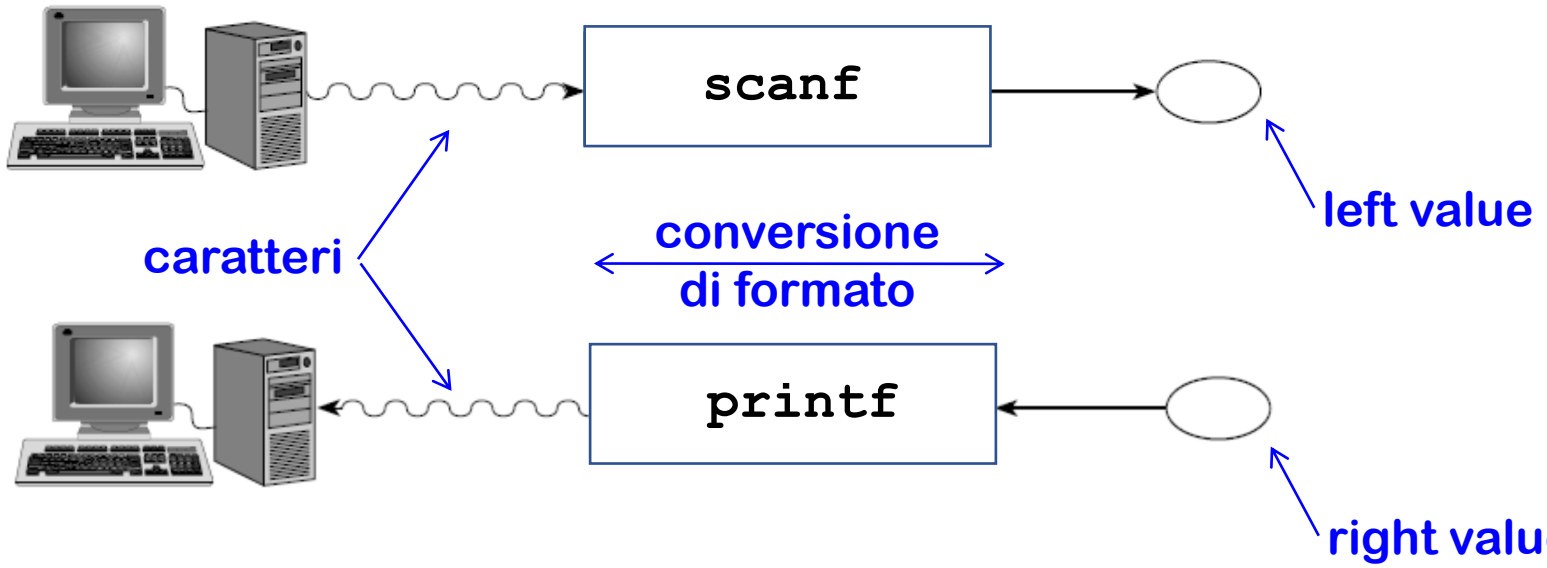
- Con le operazioni di **output**, un'espressione viene valutata ed il valore ottenuto viene presentato sull'unità di uscita (schermo).

Operazioni di Input/Output

- Il C usa un'utile astrazione (i **flussi** o **streams**) per realizzare le operazioni di I/O con dispositivi come la tastiera e lo schermo.
- Uno stream è un oggetto dove un programma può inserire o estrarre caratteri e che virtualizza i dispositivi fisici ad esso associati.
- La libreria standard C include il file header **`stdio.h`** dove sono dichiarate le funzioni per input e output.

Operazioni di Input/Output

- Gli stream gestiscono flussi di caratteri
- L'operazione di input avviene su una variabile (left value)
- L'operazione di output avviene su un'espressione (right value)



Operazione di output: **`printf`**

`printf``(`*`stringa di formato, altri argomenti`*`)`

- Stampa sullo standard output i caratteri contenuti nella *stringa di formato*.
- Se la stringa contiene specificatori di formato (sequenze di caratteri inizianti con %), gli argomenti addizionali sono *formattati* e inseriti nella stringa in output al posto dei loro rispettivi specificatori.
- Il numero degli argomenti deve essere uguale a quello degli specificatori.
- Per utilizzare **`printf`** bisogna introdurre **`#include <stdio.h>`**

Operazione di output: **`printf`**

- Alcuni specificatori di formato:
- **`%d`** per gli interi (**`int`**)
- **`%f`** per i numeri reali (**`float`** e **`double`**)
- **`%c`** per i caratteri singoli (**`char`**)
- **`%s`** per le stringhe di caratteri

Esempio

```
#include <stdio.h>
int main() {
  int p;
  p = 18;
  printf("Io ho %d anni\n", p);
}
```

Operazione di input: **`scanf`**

`scanf`(*`stringa di formato`*,*`elenco variabili`*)

- La funzione **`scanf`** legge caratteri dallo stream di input, li converte in dati di un certo tipo in accordo a quanto precisato dalla *stringa di formato*. I dati sono poi memorizzati all'interno delle variabili presenti *nell'elenco variabili*.

Operazione di input: **`scanf`**

- La *stringa di formato* è simile a quanto visto per **printf**
- Contiene:
  - ***Caratteri blank***, in corrispondenza dei quali la funzione leggerà e salterà tutti i caratteri blank che incontra in input prima del primo carattere non-blank. Sono caratteri blank lo spazio, il newline ed il carattere di tabulazione;
  - ***Specificatori di formato***: una sequenza formata da un carattere iniziale %, usata per specificare il tipo ed il formato del dato da leggere in input e da assegnare alla variabile corrispondente.
- Ci dovranno essere tanti specificatori di formato quante sono le variabili nell'elenco.

Operazione di input: `scanf`

- Alcuni specificatori di formato:
- `%d` – legge una sequenza di caratteri costituita da cifre decimali (0-9), opzionalmente preceduta da un segno (+ o -). La lettura della sequenza termina al primo carattere blank. Il dato viene memorizzato come un intero nella variabile corrispondente
- `%c` – legge il prossimo carattere e lo salva nella variabile corrispondente
- `%f` - legge una sequenza di caratteri costituita da cifre decimali (0-9), opzionalmente contenente un punto decimale, opzionalmente preceduta da un segno (+ o -). La lettura della sequenza termina al primo carattere blank. Il dato viene memorizzato come un reale nella variabile corrispondente

Operazione di input: **`scanf`**

- L'elenco delle variabili contiene gli identificatori delle variabili che saranno assegnate dalla funzione **`scanf`**
- Più precisamente, di ogni variabile sarà fornito l'indirizzo di memoria nel quale essa è memorizzata. A questo scopo, si utilizza l'operatore & che viene prefisso ad ogni variabile da modificare.
- Esempio (lettura da input di due variabili intere):

```
int a,b;
scanf("%d %d", &a, &b);
```

Strutture di controllo

- Ricordiamo che ogni algoritmo può essere implementato impiegando solo tre tipi di strutture per il controllo di flusso:
  - Sequenza
  - Selezione
  - Ciclo
- Tipicamente i linguaggi offrono più di un costrutto per ogni tipo di struttura per rendere più agevole la codifica degli algoritmi

Sequenza

- La sequenza è costituita da un insieme di istruzioni successive che vengono eseguite nell'ordine in cui compaiono nel testo.
- Un particolare caso di sequenza in C è il **blocco** (o *istruzione composta*, *compound statement*), formato da un insieme di istruzioni tra parentesi graffe { }.

Costrutti di selezione

- Permettono di scegliere di eseguire una tra due istruzioni alternative in base alla valutazione di una espressione logica
- Tre principali costrutti
  - **`if`**
  - **`if … else`**
  - **`if … else if … else`**

Costrutti di selezione: `if`

Sintassi

```
if(condizione)
  istruzione
```

- L'*istruzione* è eseguita solo se *condizione* è **true**
- L'*istruzione* può essere costituita da un blocco.

Esempi

- Calcolo del valore assoluto di un numero dato in input
- Verificare che due valori X e Y forniti in input rispettino la condizione X >= Y.

Costrutti di selezione: **`if…else`**

Sintassi

```
if(condizione)
  istruzione_1
else
  istruzione_2
```

- L'*`istruzione_1`* è eseguita solo se *`condizione`* è **`true`**
- L'*`istruzione_2`* è eseguita solo se *`condizione`* è **`false`**
- *`istruzione_1`* e *`istruzione_2`* possono essere costituite da blocchi

Esempio: max fra due

```
# include <stdio.h>

int main()
{
  int x,y,max;

  printf("Primo valore: ");
  scanf("%d",&x);
  printf("Secondo valore: ");
  scanf("%d",&y);

  if(x>y)
    max=x;
  else
    max=y;

  printf("Il massimo tra %d e %d e': %d\n", x, y, max);
}
```

Esempio: max fra tre

```
# include <stdio.h>

int main()
{
  int x,y,z,max;

  printf("Primo valore: ");
  scanf("%d",&x);
  printf("Secondo valore: ");
  scanf("%d",&y);
  printf("Terzo valore: ");
  scanf("%d",&z);

  max=x;

  if(y>max)
    max=y;

  if(z>max)
    max=z;

  printf("Il massimo e': %d\n", max);
}
```

Costrutti di selezione: **`if…else if … else`**

Sintassi

```
if(condizione_1)
  istruzione_1
else if(condizione_2)
  istruzione_2
else if(condizione_3)
  istruzione_3
else
  istruzione_4
```

- L'*`istruzione_1`* è eseguita solo se *`condizione_1`* è **`true`**
- L'*`istruzione_2`* è eseguita solo se *`condizione_1`* è **`false`** e *`condizione_2`* è **`true`**
- L'*`istruzione_3`* è eseguita solo se *`condizione_1`* è **`false`**, *`condizione_2`* è **`false`** e *`condizione_3`* è **`true`**
- L'*`istruzione_4`* è eseguita solo se *`condizione_1`* è **`false`**, *`condizione_2`* è **`false`** e *`condizione_3`* è **`false`**
- *`istruzione_1`* e *`istruzione_2`* possono essere costituite da blocchi

Esempio

```
# include <stdio.h>

int main()
{
  int voto;

  printf("Voto ricevuto: ");
  scanf("%d", &voto);

  if (voto < 18)
   printf(" Ritorna\n");
  else if (voto < 24)
    printf(" Si puo' dare di piu'! \n ");
  else if (voto < 27)
    printf(" Non c'e' male !\n ");
  else if (voto < 30)
    printf("Bene!\n");
  else if (voto == 30)
    printf(" Finalmente ci siamo ! \n");
  else
    printf(" WOW!!! \n");
}
```

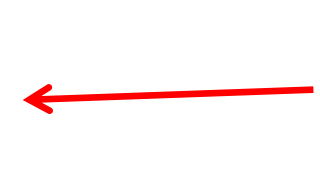
Costrutti di selezione: **`switch`**

- Viene valutata *espr_sw*
- Il valore ottenuto viene confrontato con le costanti *const_i* contenute nei **case**
- In corrispondenza della prima uguaglianza, si eseguono le istruzioni relative
- Se nessuna uguaglianza è verificata ed è presente l'etichetta **default**, vengono eseguite le istruzioni relative

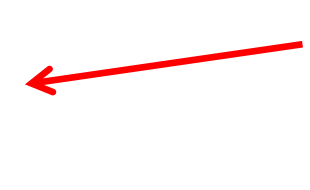
```
switch(espr_sw){
case const_1:
   istruzione_1_1
    ...
    istruzione_1_m
    break;
case const_2:
    istruzione_2_1
    ...
    istruzione_2_n
    break;
case const_3: case const_4:
    istruzione_3_1
    ...
    istruzione_3_t
    break;
default:
    istruzione_3_1
    ...
    istruzione_3_t
    break;
}
```

Costrutti di selezione: **switch**

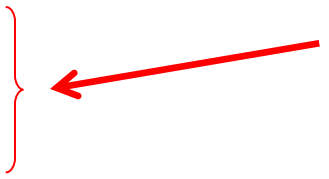
```
switch(espr_sw){
case const_1:
   istruzione_1_1
    ...
    istruzione_1_m
    break;
case const_2:
    istruzione_2_1
    ...
    istruzione_2_n
    break;
case const_3: case const_4:
    istruzione_3_1
    ...
    istruzione_3_t
    break;
default:
    istruzione_3_1
    ...
    istruzione_3_t
    break;
}
```



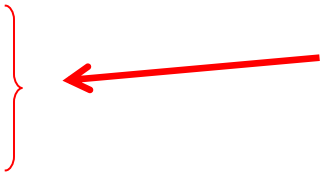
eseguite se *espr_sw==const_1*



eseguite se *espr_sw!= const_1* e *espr_sw== const_2*



eseguite se *espr_sw!= const_1*, *espr_sw!= const_2* e *espr_sw== const_3* o *espr_sw== const_4*



eseguite se *espr_sw* è diverso da tutte le *const_i* nei **case**

Costrutti di ciclo

- Servono a ripetere l'esecuzione di un'istruzione
- A seconda di come viene definito il numero di ripetizioni dell'esecuzione, si distinguono in
  - **Costrutti di ciclo a condizione**
  - **Costrutti di ciclo a conteggio**

Costrutti di ciclo: **`while`**

- E’ un costrutto di ciclo **a condizione iniziale**
- Non si definisce esplicitamente il numero di ripetizioni dell’esecuzione, ma si valuta all’inizio del ciclo un’espressione logica che, fin quando risulta vera, causa un’ulteriore esecuzione dell’istruzione.

Costrutti di ciclo: **`while`**

Sintassi

`while`(*`condizione`*)
  `istruzione`

- Si valuta la *`condizione`*
- Se risulta vera, si esegue l'istruzione e quindi si torna a verificare la condizione
- Se la condizione risulta falsa, si passa a eseguire le istruzioni che si trovano dopo la chiusura del **`while`**
- Qual è il minor numero di cicli che si può effettuare ?

Costrutti di ciclo: **`do...while`**

- E' un costrutto di ciclo **a condizione finale**
- Non si definisce esplicitamente il numero di ripetizioni dell'esecuzione, ma si valuta al termine del ciclo un'espressione logica che, fin quando risulta vera, causa un'ulteriore esecuzione dell'istruzione.

Costrutti di ciclo: **do…while**

Sintassi

```
do
    istruzione
while(condizione);
```

- Si esegue l'*istruzione*
- Si valuta la *condizione*
- Se risulta vera, si torna ad eseguire l'*istruzione*
- Se la condizione risulta falsa, si passa a eseguire le istruzioni che si trovano dopo la chiusura del **while**
- Qual è il minor numero di cicli che si può effettuare ?

Costrutti di ciclo: `for`

- E’ un costrutto di ciclo **a conteggio**
- Si definisce esplicitamente il numero di ripetizioni dell’esecuzione
- Il conteggio viene gestito grazie ad una variabile (*variabile di conteggio*) che assume un valore iniziale e viene incrementata di un valore fisso ad ogni ripetizione del ciclo finché non raggiunge o supera un valore finale.

Costrutti di ciclo: **`for`**

Sintassi

```
for(inizialization; condition; increase)
  istruzione
```

- Si esegue *`inizialization`*
- Si valuta *`condition`*
- Se risulta vera, si esegue l'*`istruzione;`* al termine dell'esecuzione, si esegue *`increase`* e si torna a valutare *`condition`*
- Se la condizione risulta falsa, si passa a eseguire le istruzioni che si trovano dopo la chiusura del **`for`**

Definizione di un array

- Per definire una variabile array, è necessario specificare:
  - il nome della variabile array
  - il tipo degli elementi
  - il numero degli elementi presenti (cardinalità dell'array)

Esempio

- Definizione di una variabile array v contenente 20 interi:

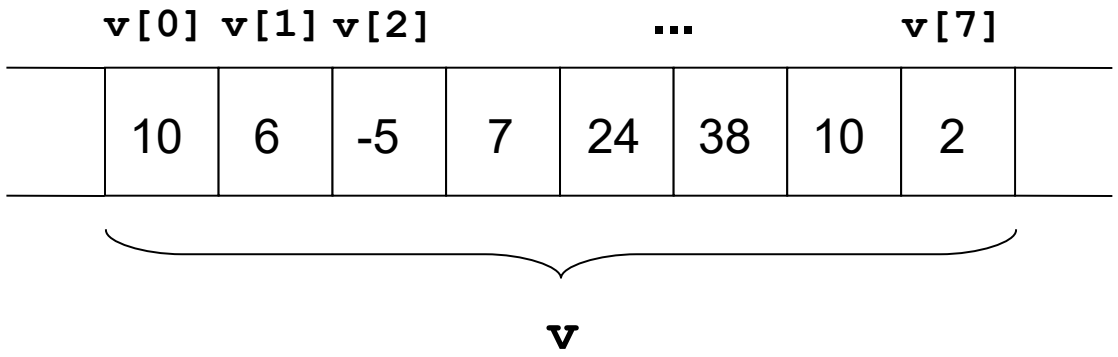
```
int v[20];
```

- Definizione di una variabile array **w** contenente 10 float:

```
float w[10];
```

Accesso agli elementi dell'array

- Per accedere ai singoli elementi di un array, è necessario specificare il nome della variabile array e la posizione dell'elemento di interesse tramite un valore intero (variabile o costante) che si definisce indice.



Accesso agli elementi dell'array

- Si noti che l'indice parte da 0; quindi `v[0]` sarà il primo valore dell'array, mentre l'N-mo sarà `v[N-1]`.
- Va quindi ricordato che, se si definisce un array con `N` elementi, l'indice dovrà essere limitato tra `0` ed `N-1`.
- Questo controllo è a cura dell'utente, in quanto non ci sono controlli automatici della correttezza dell'indice. Nel caso si consideri un indice errato (es. `v[N]`), sarà effettuato un accesso ad una zone della memoria che non appartiene all'array, con effetti imprevedibili a runtime.

Accesso agli elementi dell'array

Ogni elemento di un array è, a tutti gli effetti, una variabile del tipo costituente l'array e quindi può essere impiegato come tale:

```
int a,b,i;
int v[10];


v[2]=3;
v[7]=0;
printf("Valore: %d\n",v[7]);
i=2;
a=v[i]*4+6;
b=v[i+5];
```

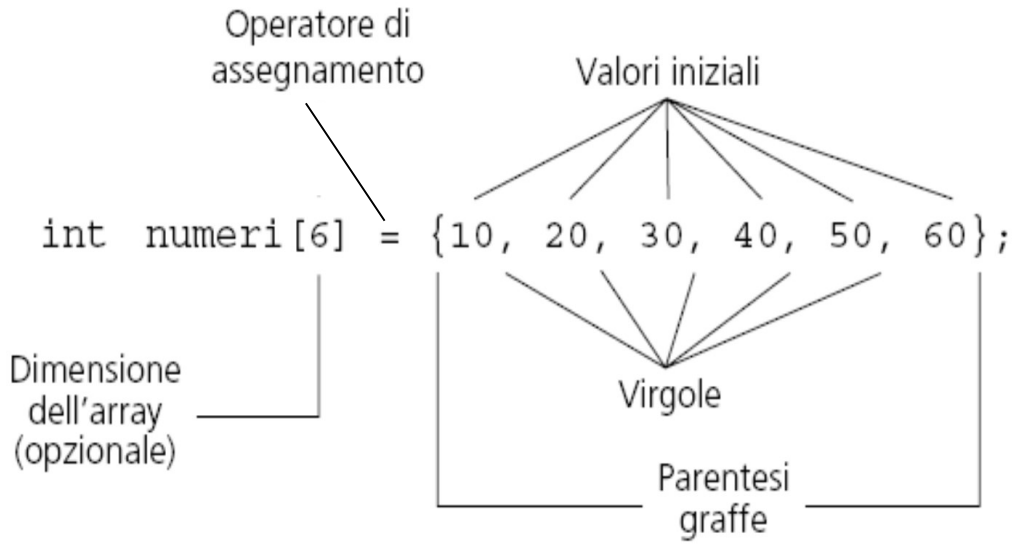
Inizializzazione di un array

- Un array può essere inizializzato in fase di definizione:

```
int numeri[6] = {10,20,30,40,50,60};
```

- La dimensione dell'array può essere anche implicita:

```
int numeri[] = {10,20,30,40,50,60};
```



Assegnazione tra array

- Diversamente dalle variabili di tipo atomico, non è possibile fare assegnazioni dirette tra array.
- L'unica possibilità per assegnare i valori degli elementi di un array agli elementi di un altro array è quindi fare una serie di assegnazioni tra elementi corrispondenti:

```
int a[]={7,9,6,3};
int b[4];

b=a;
```

errata

```
b[0]=a[0];
b[1]=a[1];
b[2]=a[2];
b[3]=a[3];
```

corretta

Dimensione dell'array

- La dimensione fornita all'atto della definizione dell'array **deve essere una costante**.
- Esempi di definizioni corrette:

```
/* dimensione costante numerica */
int vet[100];

/* dimensione costante definita */
#define SIZE 150
…
int vet[SIZE];
```

Dimensione dell'array

- Nel caso il numero degli elementi da memorizzare in un array fosse noto solo a tempo di esecuzione, comunque la dimensione dell'array da impiegare **deve essere una costante**
- In questo caso, per la definizione si sceglie una dimensione che si ritiene adeguata ad ospitare il numero massimo di elementi previsto e si affianca all'array una variabile intera che contiene il numero effettivo di elementi memorizzati nell'array (riempimento).

Lettura e stampa degli elementi di un array

- Per inizializzare da input una variabile array, è necessario realizzare un'operazione di input per ciascuno degli elementi
- Analogamente, per stampare il contenuto di un array, è necessario fare la stampa di ognuno degli elementi.
- Qual è il costrutto da utilizzare ?
- Problema:
  - leggere da input la dimensione e gli elementi di un array e stampare il risultato della lettura

```c
#include <stdio.h>
#define MAXSIZE 100
int main() {
        int vet[MAXSIZE];
        int i,num;

        printf("Quanti elementi?");
        scanf("%d",&num);
        while(num > MAXSIZE){
                        printf("Numero eccessivo!\n");
                        printf("Quanti elementi?");
                        scanf("%d", &num);
        }

        for(i=0; i < num; i++){
                        printf("Fornire il valore n.%d:",i)
                        scanf("%d",&vet[i]);
        }

        for(i=0; i < num; i++)
                        if(vet[i] < 0)
                                vet[i] = -vet[i];

        for(i=0; i < num; i++)
                        printf("Valore [%d]: %d\n",i, vet[i]);
}
```

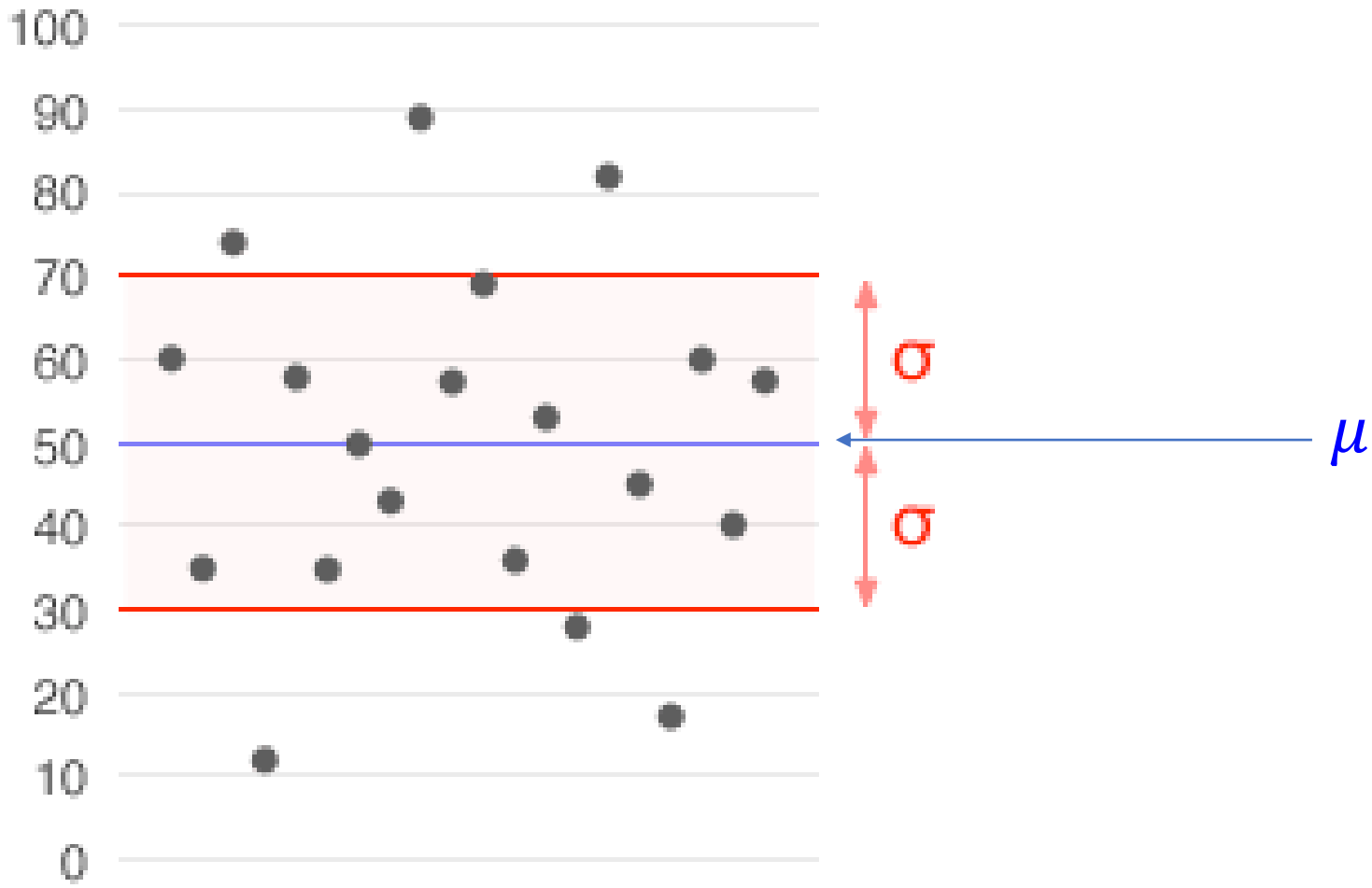
```c
#include <stdio.h>
#define MAXSIZE 100
int main() {
        int vet[MAXSIZE];
        int i,num;

        printf("Quanti elementi?");
        scanf("%d",&num);
        while(num > MAXSIZE){
                        printf("Numero eccessivo!\n");
                        printf("Quanti elementi?");
                        scanf("%d", &num);
        }

        for(i=0; i < num; i++){
                        printf("Fornire il valore n.%d:",i)
                        scanf("%d",&vet[i]);
        }

        for(i=0; i < num; i++)
                        if(vet[i] < 0)
                                vet[i] = -vet[i];

        for(i=0; i < num; i++)
                        printf("Valore [%d]: %d\n",i, vet[i]);
}
```

Uso della costante definita

Uso del riempimento

Problema:
calcolo della media e della deviazione standard di un insieme di dati

Ogni punto sul diagramma rappresenta un particolare dato, il cui valore può essere letto sulla scala a sinistra.
La linea blu indica il valore medio (μ) dei dati.
La linea rossa indica la deviazione standard (σ) dei dati .



Problema:
calcolo della media e della deviazione standard di un insieme di dati

- Leggere da input la dimensione *n* e gli elementi di un array di numeri reali; fornire in uscita la media μ e la deviazione standard σ degli elementi presenti nell'array, dove:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}$$

x_i ←→ `x[i]`

Problema (versione 2): calcolo della media e della deviazione standard di un insieme di dati

- Nelle stesse ipotesi dell'esercizio precedente, calcolare la media μ e la deviazione standard σ di un insieme di valori reali letti in un array, dove stavolta si usa l'espressione:

$$\sigma = \frac{1}{n}\sqrt{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}$$

- Che cosa è cambiato? È cambiato in meglio o in peggio?

Definizione di un array bidimensionale

- Per definire un array bidimensionale, è necessario specificare:
  - il **nome** della variabile array
  - il **tipo** degli elementi
  - il **numero degli elementi presenti nelle due dimensioni** (cardinalità di riga e cardinalità di colonna dell'array)

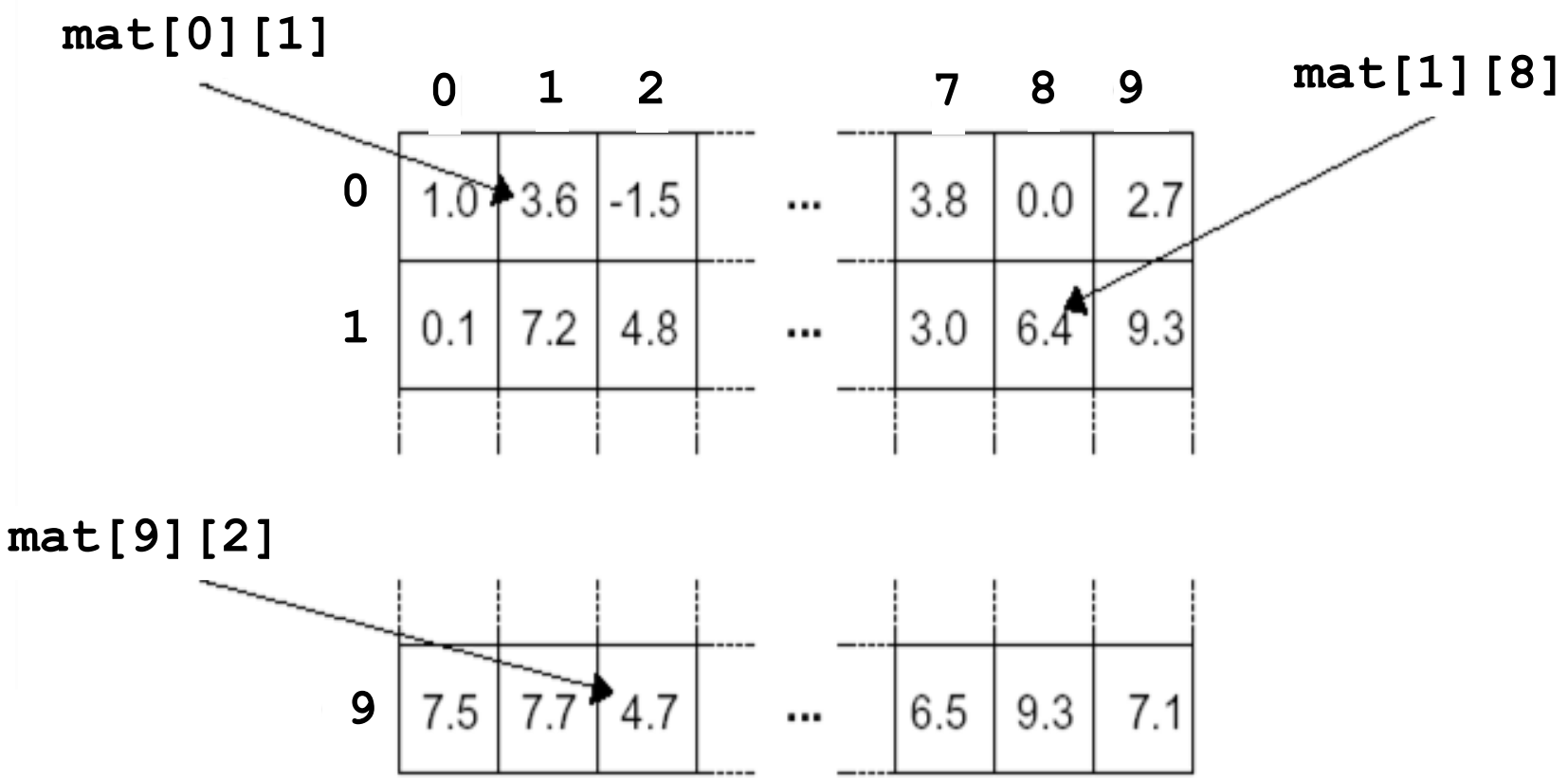
Esempio

- Definizione di una variabile array **`mat`** contenente 10x10 elementi **`double`**:

```
double mat[10][10];
```

- **Che differenza c'è rispetto ad un array monodimensionale di 100 elementi?**

Organizzazione di un array bidimensionale



Accesso agli elementi dell'array bidimensionale

- Per accedere ai singoli elementi di un array bidimensionale, è necessario specificare il nome della variabile array e gli indici di riga e di colonna che individuano l'elemento desiderato.
- **Esempi**:

```
mat[2][1]=3;
printf("il valore è: %d\n", mat[2][7]);
i=3;
j=5;
x=mat[i][j]*4+6;
```

Inizializzazione di un array bidimensionale

- Un array bidimensionale può essere inizializzato in fase di definizione:

```
int mat[2][3] = {{10,20,30},{40,50,60}};
```

- Altre inizializzazioni equivalenti:

```
int mat[2][3] = {10,20,30,40,50,60};

int mat[][3] = {{10,20,30},{40,50,60}};
```

Lettura e stampa di un array bidimensionale

- Anche nel caso bidimensionale, l'inizializzazione da input di una variabile array va realizzata realizzare tramite un'operazione di input per ciascuno degli elementi
- Analogamente, per stampare il contenuto di un array, è necessario fare la stampa di ognuno degli elementi.
- **Qual è il costrutto da utilizzare ?**
- Esempio:
  - leggere da input le dimensioni e gli elementi di un array bidimensionale e stampare il risultato della lettura

Array multidimensionali

- Il C permette la definizione di array multidimensionali con più di due indici. Esempio:
  ```
  int mat3[5][10][5];
  ```

- Con le dovute modifiche valgono le considerazioni sulla definizione, inizializzazione, assegnazione, accesso fatte per gli array monodimensionali e bidimensionali.