ROMA TRE
UNIVERSITÀ DEGLI STUDI

Dipartimento di Ingegneria
Industriale, Elettronica e Meccanica

Collegio Didattico di Ingegneria Meccanica

Elementi di Informatica

Array – Parte I

Giordano Da Lozzo e Giuseppe Sansonetti

Programma per trovare il minimo

Vogliamo scrivere un programma che prima legga una sequenza di interi e poi ne calcoli il minimo.

Minimo fra due interi

```c
/* programma che prima legge due interi e poi ne calcola il minimo */
int main() {

    int x1, x2; // interi da leggere
    int minimo; // risultato

    /* input */
    printf("Caro utente introduci due interi\n");
    scanf("%d", &x1);
    scanf("%d", &x2);

    /* calcola e stampa risultato */
    minimo = x1;
    if(x2<minimo)
        minimo = x2;

    printf("Il minimo vale %d", minimo);
}
```

Minimo fra tre interi

```c
/* programma che prima legge tre interi e poi ne calcola il minimo */
int main() {

    int x1, x2, x3; // interi da leggere
    int minimo;     // risultato

    /* input */
    printf("Caro utente introduci tre interi\n");
    scanf("%d", &x1);
    scanf("%d", &x2);
    scanf("%d", &x3);

    /* calcola e stampa risultato */
    minimo = x1;
    if(x2<minimo)
        minimo = x2;
    if(x3<minimo)
        minimo = x3;

    printf("il minimo vale %d", minimo);
}
```

Minimo fra 20 interi

```
/* programma che prima legge venti interi e poi ne calcola il minimo */

int main() {
    int x1, x2, x3, x4, x5, x6, x7, x8, x9, x10;          // interi da leggere
    int x11, x12, x13, x14, x15, x16, x17, x18, x19, x20; // interi da leggere

    int minimo; // risultato
```

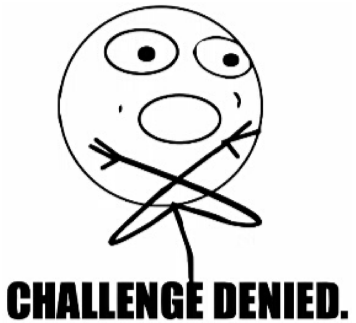
```c
/* input */
printf("Caro utente introduci venti interi\n");
scanf("%d", &x1);
scanf("%d", &x2);
scanf("%d", &x3);
scanf("%d", &x4);
scanf("%d", &x5);
scanf("%d", &x6);
scanf("%d", &x7);
scanf("%d", &x8);
scanf("%d", &x9);
scanf("%d", &x10);
scanf("%d", &x11);
scanf("%d", &x12);
scanf("%d", &x13);
scanf("%d", &x14);
scanf("%d", &x15);
scanf("%d", &x16);
scanf("%d", &x17);
scanf("%d", &x18);
scanf("%d", &x19);
scanf("%d", &x20);
```

```
/* calcola e stampa risultato */
minimo = x1;
if(x2<minimo)
   minimo = x2;
if(x3<minimo)
   minimo = x3;
if(x4<minimo)
   minimo = x4;
if(x5<minimo)
   minimo = x5;
if(x6<minimo)
   minimo = x6;
if(x7<minimo)
   minimo = x7;
if(x8<minimo)
   minimo = x8;
if(x9<minimo)
   minimo = x9;
if(x10<minimo)
   minimo = x10;
if(x11<minimo)
   minimo = x11;
if(x12<minimo)
   minimo = x12;
if(x13<minimo)
   minimo = x13;
if(x14<minimo)
   minimo = x14;
if(x15<minimo)
   minimo = x15;
if(x16<minimo)
   minimo = x16;
if(x17<minimo)
   minimo = x17;
if(x18<minimo)
   minimo = x18;
if(x19<minimo)
   minimo = x19;
if(x20<minimo)
   minimo = x20;

   printf("Il minimo vale %d", minimo);
}
```

Limiti della soluzione proposta

- La lunghezza del codice cresce al crescere del numero di interi.
  Prova a scrivere un programma che legge 1000 interi e poi ne calcola il minimo.



- Bisogna scrivere programmi diversi per problemi molto simili.
  Il programma per calcolare il minimo tra 20 interi contiene variabili e istruzioni diverse rispetto al programma per calcolare il minimo tra 3 interi.

- Non è possibile gestire la situazione in cui il numero di interi non è noto a priori. Ovvero non è possibile scrivere un programma che calcola il minimo fra *n* interi con l'approccio proposto.

Caratteristiche del problema

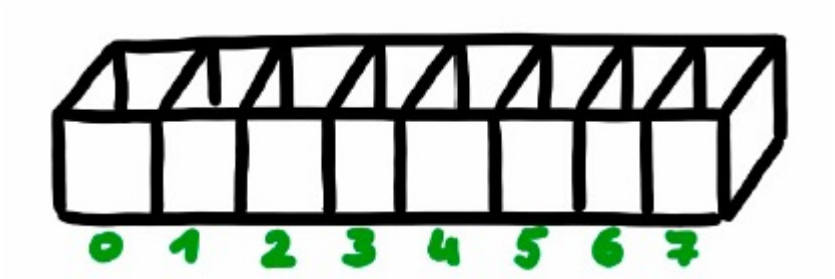
Abbiamo la necessità di gestire un insieme di variabili omogenee, ovvero:

1. dello stesso tipo.
   - Es: nel programma per il calcolo del minimo dobbiamo gestire un insieme di variabili di tipo int.
2. con lo stesso significato, ovvero tutte le variabili rappresentano lo stesso tipo di informazione, su cui vanno eseguite le stesse operazioni.
   - Es: nel programma per il calcolo del minimo, a ciascuna variabile va assegnato un valore letto da tastiera, ciascuna variabile va poi acceduta per confrontarne il valore con il valore della variabile *minimo*, e va di nuovo eventualmente acceduta per memorizzarne il valore *minimo*.

Array

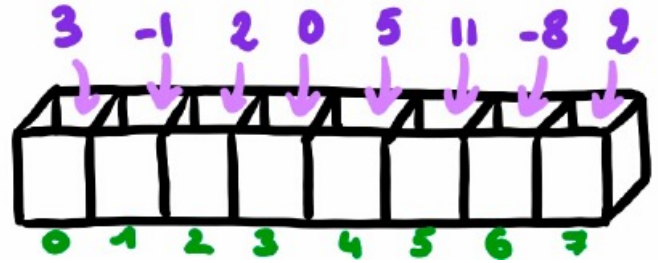
E’ una sequenza di variabili omogenee.

Permette di accedere in lettura ed in scrittura a ciascuna delle variabili che lo compongono sulla base della loro posizione, detta indice.



Array

E’ una sequenza finita di variabili dello stesso tipo.



Array composto da
8 variabili di tipo int

Elementi: le variabili che compongono un array prendono il nome di elementi dell’array.

Sequenza: gli elementi dell’array sono ordinati, quindi si può parlare del primo elemento, del secondo elemento, …

Sequenza finita: il numero di variabili che compongono l’array è un numero finito, detto lunghezza o dimensione dell’array.

Indice: numero intero non negativo associato a ciascun elemento, ne indica la posizione. Le posizioni sono 0, 1, 2, …, lunghezza-1.

Array: concetti ulteriori

Gli array (monodimensionali) sono anche detti vettori.

Gli array costituiscono una struttura dati, ovvero un'entità o un metodo utilizzato per memorizzare insiemi di dati.

- Il metodo per memorizzare dati associato ad un array è quello di memorizzare i valori degli elementi uno dopo l'altro (logicamente e vedremo anche fisicamente), nell'ordine in cui compaiono nell'array.

Gli array sono variabili strutturate, mentre le variabili semplici sono non strutturate.

Gli array hanno un tipo, che è il tipo dei loro elementi.

Dichiarazione di array

Per usare un array, è necessario dichiarare una variabile che permetta di referenziare l'array, specificando tipo, nome e lunghezza dell'array.

Esempi: int interi[20]; float reali[15/3];

Sintassi: tipo nome[lunghezza];

Semantica: alloca un'area di memoria sufficiente a contenere l'array.

Nota: lunghezza deve essere una espressione di tipo int

```
int interi[3.5];
```

```
error: size of array 'interi' has non-integer type
```

Dichiarazione di array

```c
#include <stdio.h>

int main(){
    int interi[20];
    float reali[20];
    char caratteri[20];
    printf("Interi occupa %d byte\n", sizeof(interi));
    printf("Reali occupa %d byte\n", sizeof(reali));
    printf("Caratteri occupa %d byte\n", sizeof(caratteri));
}
```

```
-bash                                    ⌥⌘1
└──>> ./allocazione
Interi occupa 80 byte
Reali occupa 80 byte
Caratteri occupa 20 byte
```

Accesso

Dopo aver dichiarato una variabile per referenziare l'array, è possibile utilizzare l'array.

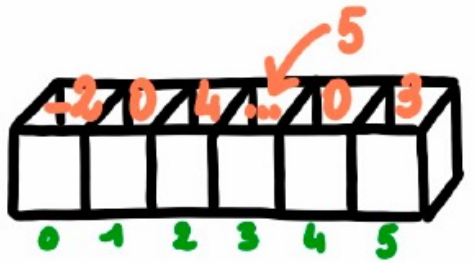
- L'unico utilizzo che si può fare di un array è accedere ai suoi elementi per memorizzare un valore al loro interno (accesso in scrittura) o per recuperare un valore già memorizzato al loro interno (accesso in lettura).

Sintassi: *nome*[*indice*]

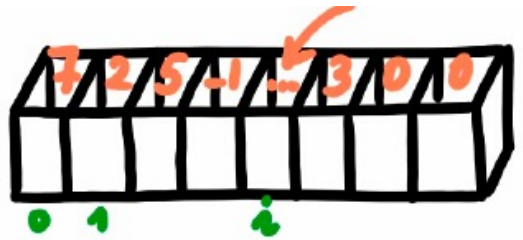
Semantica: accede all'elemento con indice *indice* all'interno dell'array referenziato da *nome*.

Accesso: esempi

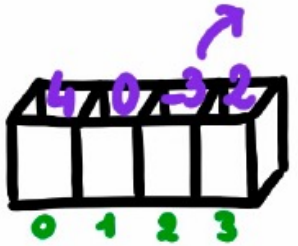
interi[3]=5;
Accesso in scrittura: memorizza all'interno dell'elemento il cui indice è 3 il valore 5;



scanf("%d", &interi[i])
Accesso in scrittura: memorizza all'interno dell'elemento il cui indice è i il valore letto da tastiera.



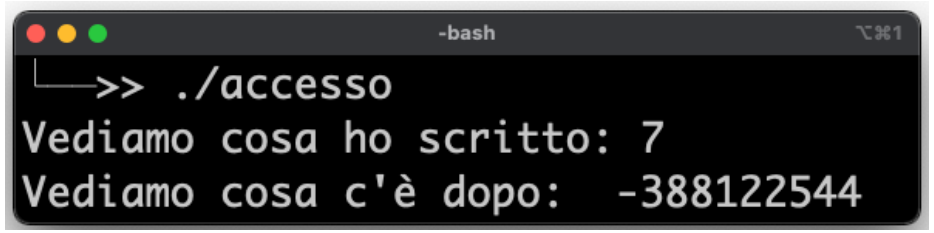
printf("%d", interi[2]);
Accesso in lettura: legge e stampa il valore (-3) dell'elemento con indice 2.



Accesso: esempi

```c
#include <stdio.h>

int main(){
    int interi[20];
    interi[3] = 7;
    printf("Vediamo cosa ho scritto: %d\n", interi[3]);
    printf("Vediamo cosa c'è dopo: %d\n", interi[4]);
}
```



Esempio: minimo fra 20 interi

```c
#include <stdio.h>

/* programma che prima legge venti interi e poi ne calcola il minimo */
int main() {

    int interi[20]; // interi da leggere
    int minimo; // risultato
    /* input */

    printf("Caro utente introduci venti interi\n");
    for(int i=0; i<20; i++)
        scanf("%d", &interi[i]);

    /* calcola e stampa risultato */
    minimo = interi[0];
    for(int i=1; i<20; i++)
        if(interi[i]<minimo)
                minimo = interi[i];

    printf("il minimo vale %d", minimo);
}
```

Esempio: minimo fra *n* interi

```c
#include <stdio.h>

/* programma che prima legge n interi e poi ne
   calcola il minimo */
int main() {

    int n;      // dimensione array
    int minimo; // risultato

    /* input */
    printf("Caro utente, quanti interi vuoi introdurre? ");
    scanf("%d", &n);

    /* leggi n interi */
    int interi[n]; // per gli interi da leggere
    for(int i=0; i<n; i++) {
        printf("introduci un intero: ");
        scanf("%d", &interi[i]);
    }

    /* calcola e stampa risultato */
    minimo = interi[0];
    for(int i=1; i<n; i++)
        if(interi[i]<minimo)
            minimo = interi[i];

    printf("il minimo vale %d", minimo);
}
```

Lettura elementi array di dimensione ignota

Per poter dichiarare un array è necessario conoscere la sua dimensione.

- Per questo motivo, prima di chiedere all'utente di introdurre i valori degli elementi dell'array, è necessario chiedere all'utente quanti valori ha intenzione di introdurre.

```
/* input */
printf("Caro utente, quanti interi vuoi introdurre?  ");
scanf("%d", &n);
```

Attenzione: la dichiarazione dell'array `int interi[n];` deve seguire la lettura della sua dimensione!

Array e iterazioni

Gli elementi di un array vengono molto spesso acceduti all'interno di istruzioni ripetitive.

- In un'istruzione ripetitiva una variabile contatore rappresenta l'indice di un elemento ed assume tutti i possibili valori dell'indice (ovvero da 0 alla lunghezza dell'array -1).
- Ad ogni esecuzione del corpo dell'istruzione ripetitiva viene svolta la stessa operazione su un elemento diverso dell'array.

```
for(int i=0; i<n; i++) {
    printf("Introduci un intero:  ");
    scanf("%d", &interi[i]);
}
```

⇨ Viene inserito in ciascun elemento dell'array un valore immesso dall'utente

```
for(int i=1; i<n; i++)
    if(interi[i]<minimo)
        minimo = interi[i];
```

⇨ Il valore di ciascun elemento dell'array a partire dal secondo viene confrontato con *minimo* e possibilmente assegnato a *minimo*

Dichiarazione con inizializzazione

Sintassi: *tipo nome[lunghezza] = {valore$_1$, valore$_2$, …, valore$_k$};*

Semantica: alloca un'area di memoria sufficiente a contenere l'array ed assegna ad i primi *k* elementi i valori *valore$_1$, valore$_2$, …, valore$_k$*.

```
int interi[6] = {1, 2, 3, 4, 5, 6};
```

è equivalente a

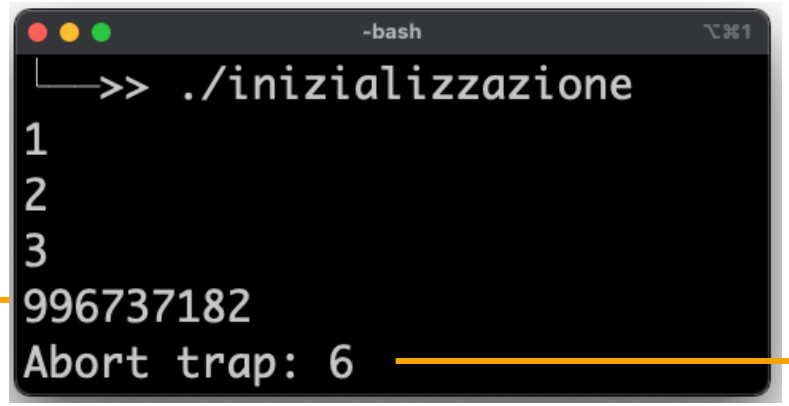
```
int interi[6];
interi[0] = 1;
interi[1] = 2;
interi[2] = 3;
interi[3] = 4;
interi[4] = 5;
interi[5] = 6;
```

Se il numero di espressioni fra graffe è uguale alla lunghezza desiderata per l'array, si può anche omettere la lunghezza nella dichiarazione con inizializzazione: `int interi[] = {1, 2, 3, 4, 5, 6};`

Dichiarazione con inizializzazione: esempio

```c
#include <stdio.h>

int main(){
    int interi[] = {1,2,3};
    printf("%d\n", interi[0]);
    printf("%d\n", interi[1]);
    printf("%d\n", interi[2]);
    printf("%d\n", interi[3]);
    interi[3] = 10;
}
```



Un accesso in lettura oltre i limiti dell'array potrebbe non causare errori a runtime (ma è concettualmente errato), a meno che non si tratti di una zona di memoria riservata (Segmentation Fault)
Ne parleremo in seguito.

Un accesso in scrittura oltre i limiti dell'array causa la terminazione del processo!

Niente assegnazione agli array

Mentre è possibile fare una dichiarazione con inizializzazione per dichiarare un array e assegnare un valore iniziale ai suoi elementi, non è possibile dichiarare un array e poi assegnare un valore all'array.

```
int main() {
    int interi[6];
    interi = {1,2,3,4,5,6};
    printf("%d", interi[0]);
}
```

```
error: expected expression before '{' token
```

```
int main() {
    int interi[6] = {1, 2, 3, 4, 5, 6};
    int interi2[6];

    interi2 = interi;
    printf("%d", interi2[0]);
}
```

```
error: assignment to expression with array type
```

Importante: una volta che hai dichiarato un array, tutto ciò che ci puoi fare è accedere ai suoi elementi.

Un array in memoria

Da un punto di vista logico, un array è una sequenza di variabili. Da un punto di vista fisico, una variabile è una porzione di memoria. Quindi, da un punto di vista fisico, un array è una sequenza di porzioni di memoria consecutive.

```
int interi[6] = {1, 2, 3, 4, 5, 6};
for(int i=0; i<=5; i++)
    printf("L'elemento interi[%d] ha indirizzo %d e occupa %d byte\n", i, &interi[i], sizeof(interi[i]));
```

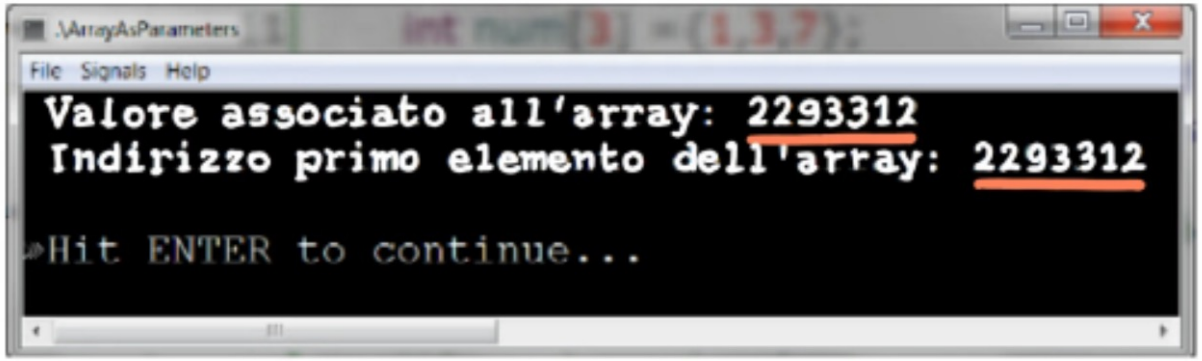
```
File  Signals  Help
L'elemento interi[0] ha indirizzo 2293296 e occupa 4 byte
L'elemento interi[1] ha indirizzo 2293300 e occupa 4 byte
L'elemento interi[2] ha indirizzo 2293304 e occupa 4 byte
L'elemento interi[3] ha indirizzo 2293308 e occupa 4 byte
L'elemento interi[4] ha indirizzo 2293312 e occupa 4 byte
L'elemento interi[5] ha indirizzo 2293316 e occupa 4 byte

Hit ENTER to continue...
```

Nome ed indirizzo

Il nome di un array è associato con un valore che è l'indirizzo di memoria del primo elemento dell'array.

```
int num[3] ={1,3,7};
printf("Valore associato al nome dell'array: %d\n", num);
printf("Indirizzo primo elemento dell'array: %d\n", &num[0]);
```

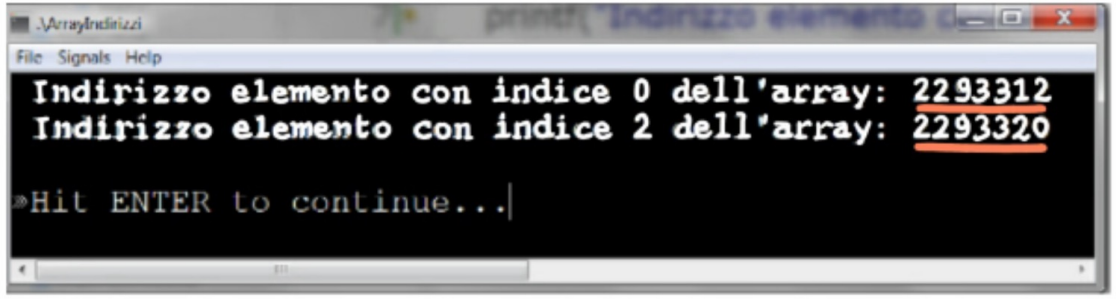


Elementi successivi al primo

Possiamo accedere all'elemento con indice *i* di un array *num* scrivendo *num[i]*.

- A livello fisico, questo equivale ad accedere alla porzione di memoria il cui indirizzo è *i*dim_elem* byte successivo a *num*, dove *dim_elem* è la dimensione di un elemento di *num*, ad esempio 4 (byte) se l'array è di tipo *int*.

```
int num[3] = {1, 3, 7};
printf("Indirizzo elemento con indice 0 dell'array: %d\n", num);.
printf("Indirizzo elemento con indice 2 dell'array: %d\n", &num[2]);
```



Elementi successivi al primo

Lunghezza e limiti di un array

Gli array – così come descritti finora – si dicono statici in quanto la loro dimensione è fissata in fase di dichiarazione e non può essere modificata.

Attenzione: provare ad accedere all'elemento di un array con indice pari alla dimensione dell'array è un errore frequente.
L'ultimo elemento di un array ha un indice pari alla *dimensione – 1*.

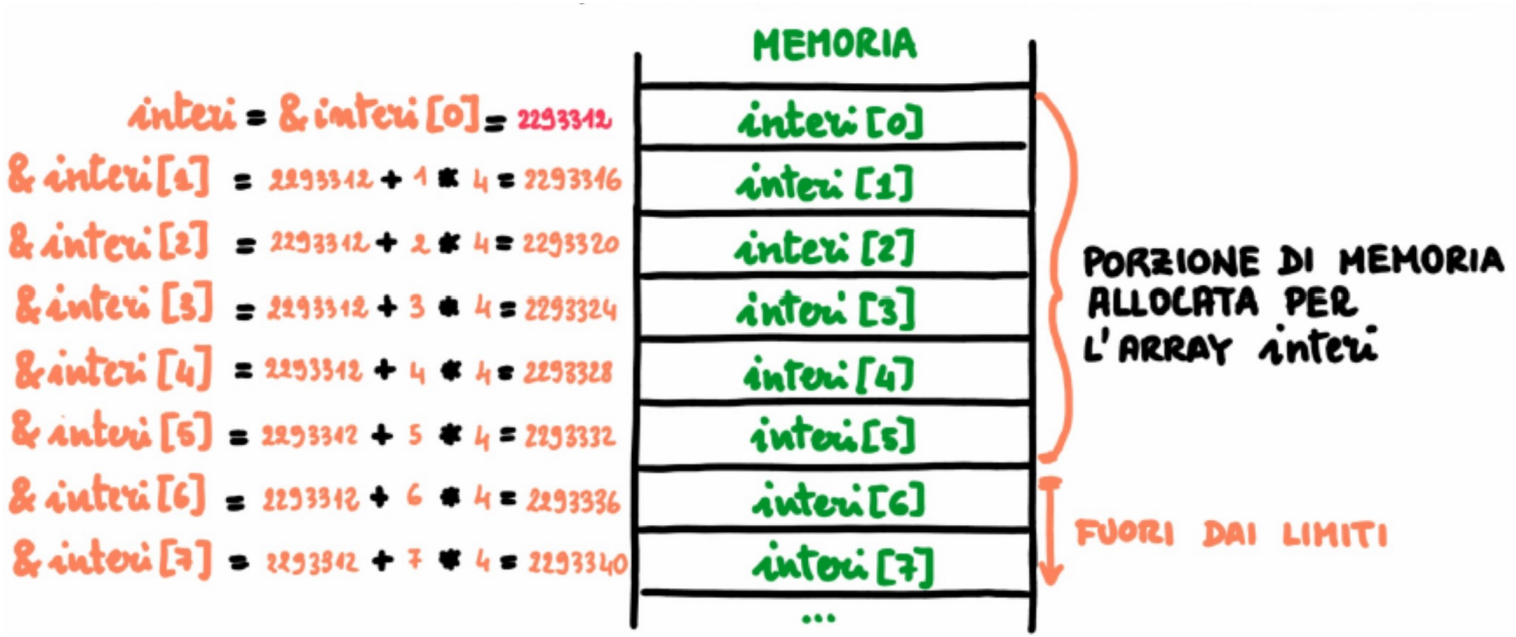
E' un errore o no?

```
int interi[6] = {1, 2, 3, 4, 5, 6};
for(int i=0; i<=6; i++)
    printf("%d ", interi[i]);
```

C non controlla i limiti di un array!

```
int interi[6] = {1, 2, 3, 4, 5, 6};
for(int i=0; i<=6; i++)
    printf("%d ", interi[i]);
```

Dopo aver stampato il valore dell'elemento con indice *5*, il programma accede alla porzione di memoria il cui indirizzo è pari all'indirizzo del primo elemento dell'array più *6*4* (ovvero, indice * la dimensione degli elementi dell'array), ovvero alla porzione di memoria di 4 byte successiva all'ultimo elemento dell'array.



Cosa succede andando fuori dai limiti?

Il comportamento di un programma quando prova ad accedere fuori dai limiti di un array non è definito:

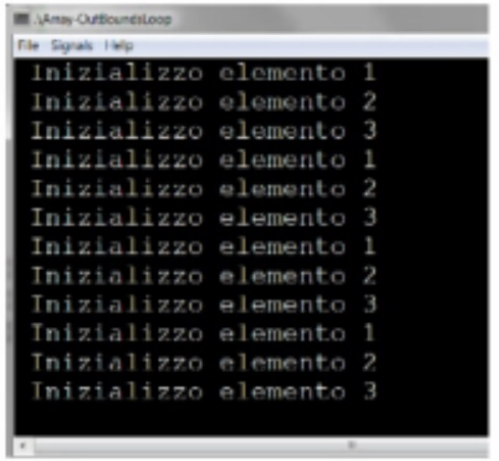
- Potrebbe non succedere nulla di sbagliato

- Potrebbe essere generato un segmentation fault, se il programma prova ad accedere ad un'area riservata

- Potrebbero essere generati degli errori logici

```
int i;
int interi[3];
for(i=0; i<=3; i++) {
    printf("Inizializzo elemento %d\n", i);
    interi[i] = 0;
}
```

Ad esempio, su sistemi Windows, questo programma genera un ciclo infinito poiché *interi[3]=0* assegna 0 alla variabile *i*

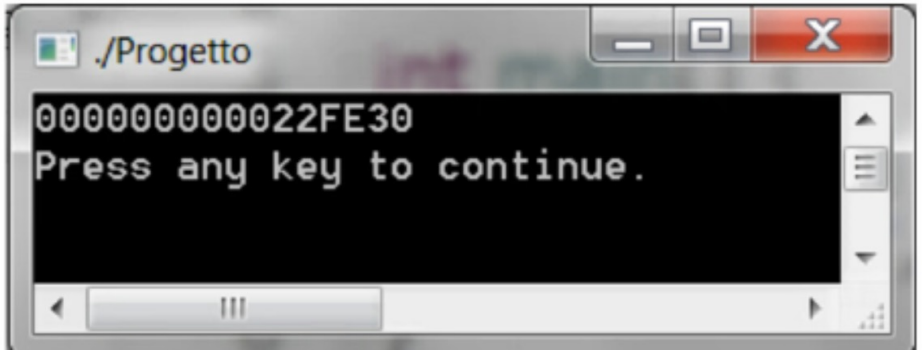


Stampa di indirizzi

In questo corso stamperemo indirizzi utilizzando il formato %d.

Questa non è la maniera più precisa: sarebbe meglio utilizzare il formato %p, dopo una conversione dell'indirizzo al tipo puntatore a *void*.

```c
int main() {
    int interi[6] = {1,2,3,4,5,6};
    printf("%p", (void*) interi);
}
```




```c
int main() {
    int interi[6] = {1,2,3,4,5,6};
    printf("%d", interi);
}
```

Altre risorse

- Bellini, Guidi: Linguaggio C — 10.1 – 10.5