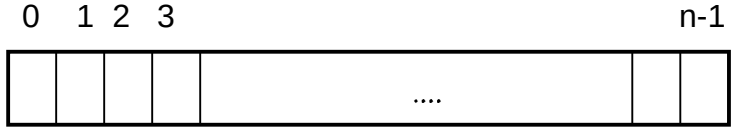
Array

Sequenza di *n* elementi adiacenti (*n* ≥ 0, costante),

omogenei (= tutti dello stesso tipo),

numerati consecutivamente da *0* a *n*-1.

Rappresentazione grafica (tipica)



n: dimens. dell'array

Dichiarazione di variabile di tipo array:

`t A[n];` dove:

- **`A`**: ident. = nome dell'array
- **`t`**: tipo (qualsiasi) = tipo degli elementi dell'array
- **`n`**: espr. intera costante = dimensione (o capacità) dell'array

Es.:

`int A[10];` variabile di nome **`A`**, di tipo array di interi, di dimensione **`10`**

`int B[20];` variabile di nome **`B`**, di tipo array di interi, di dimensione **`20`** (n.b. stesso tipo dell'array **`A`**)

`char S[32];` variabile di nome **`S`**, di tipo array di caratteri, di dimensione **`32`**

Possibile anche dichiarazione di array con inizializzazione: specifica *m* valori per gli *n* elementi dell'array ($m \leq n$). Altrimenti, il valore degli elementi è non determinato *(come per dichiarazione di variabile semplice non inizializzata)*

Es.:

```
char Vocali[5] = {'a','e','i','o','u'};
```

n.b. se $m > n$: errore (a compile-time)

se $m < n$: inizializza i primi *m* elementi dell'array, azzera gli altri (indipendentemente dal tipo).

Ad es.:

```
int A[10] = {2,-1,1}
```

i restanti 7 elementi sono posti a 0

Possibile anche

```
float F[] = {1.0,-1.5,+1.5};
```

in questo caso la dimensione dell' array F è "dedotta" dalle dimensioni dell' inizializzatore

(n.b, senza inizializzazione la dimensione deve sempre essere specificata con una costante)

Operazione di selezione

A ciascun elemento di un array

`t A[n]`

associato univocamente un numero intero tra 0 e n-1 (= indice)

L'espressione

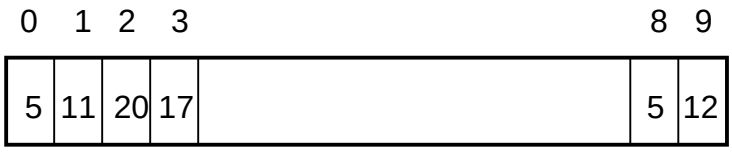
`A[e]`

con e espressione intera qualsiasi, seleziona l'elemento di A di indice *val(*e*) (n.b., prima valuta e, poi seleziona)*

Es.:

```
int A[10] = {5,11,20,17,8,4,9,13,5,12};
```

e cioè:




```
cout << A[0];    → 5
int i = 2;
cout << A[i];    → 20
cout << A[i+1];  → 17
cout << A[i]+1;  → 21
cout << A[0];    → 5
cout << A[0];    → 5
```

L'espressione ***A[…]*** *può apparire anche a destra dell'assegnamento.*

Es.:

```
int i = 2;
A[1] = 7;
A[2] = A[1] * A[0];
A[i] = A[i]+1;  (o, equiv., A[i]++)
A[i] = A[i+1];
int B[5] = {7,3,1,4,2};
cout << A[B[i]];    → 7
```

n.b. Se *val(**e**)* in **A[e]** è < 0 o $\geq n$ (*n* dimens. dell'array):

errore (logico), risultato indefinito.

Il C++ però non dà nessuna segnalazione (nè a *compile-time*, né a *run-time*)

Fare attenzione! *(in C++ possibile usare in alternativa la classe predefinita vector o definire una propria classe che preveda anche il controllo a run-time di correttezza degli indici)*

Memorizzazione di un array **`t A[n]`**:

n celle di memoria di tipo **`t`,** consecutive, a partire da un **indirizzo base** *b (n.b., indirizzo base stabilito dal sistema, al momento dell'allocazione dell'array; l'utente non ne ha nessun controllo)*

Cella di memoria di tipo **`t`** = sequenza di *m* byte consecutivi utilizzati per memorizzare un dato di tipo **`t`**; ad es., *m* = 4 byte per `int`, *m* = 1 byte per `char`)

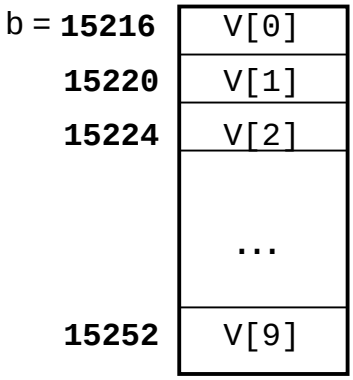
Ad es.:

```
int V[10];
```

alloca 10 celle di tipo intero = 10 x 4 = 40 byte.

Se assumiamo b = **15216** e *m* = 4:

V



n.b.: si assume che gli indirizzi di memoria siano relativi al singolo byte; quindi la cella successiva alla prima avrà indirizzo (in byte) **15216** + **4**

Ad es.:

```
char S[32];
```

alloca 32 celle di tipo carattere = 32 x 1 = 32 byte.

Con questa implementazione, facile garantire **accesso diretto**: tempo di accesso al singolo elemento indipendente dalla sua posizione *(in contra-sto con accesso sequenziale).*

Dato un array `t A[n]`

con indirizzo base *b* e dimensione (in byte) di un singolo elemento *d* (*d* = sizeof(**t**))

l'**indirizzo di memoria** del generico elemento **A[***i***]**, *i* = *0,…,n-1*, è dato da:

$$b + i * d$$

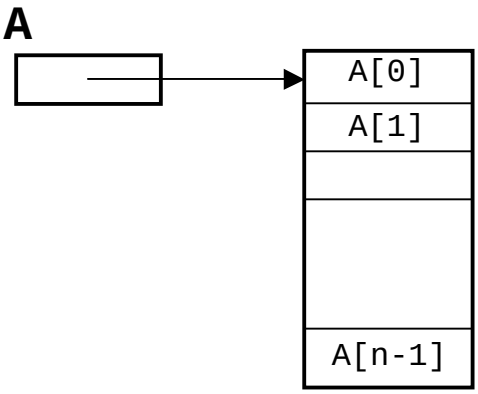
Ad es., per l'array **V** dell'esempio:

ind. di **V[3]**

→ **15216 + 3 * 4 = 15228**

n.b. Allocazione dell'array **`t A[n]`** in C++:

A è una cella di memoria che contiene l' indirizzo di base dell' array (≠**A** è un **puntatore** a `t`)



Solitamente, possibile uso array senza doverne conoscere l'implementazione concreta.

Operazioni su array

da completare

Esempio - array simmetrici

da completare

Dimensione array

da completare