

Puntatori e riferimenti (pointers and references)

Liceo G.B. Brocchi
Classi seconde Scientifico - opzione scienze applicate
Bassano del Grappa, Gennaio 2023

L-valori e R-valori

```
int x = 5;
```

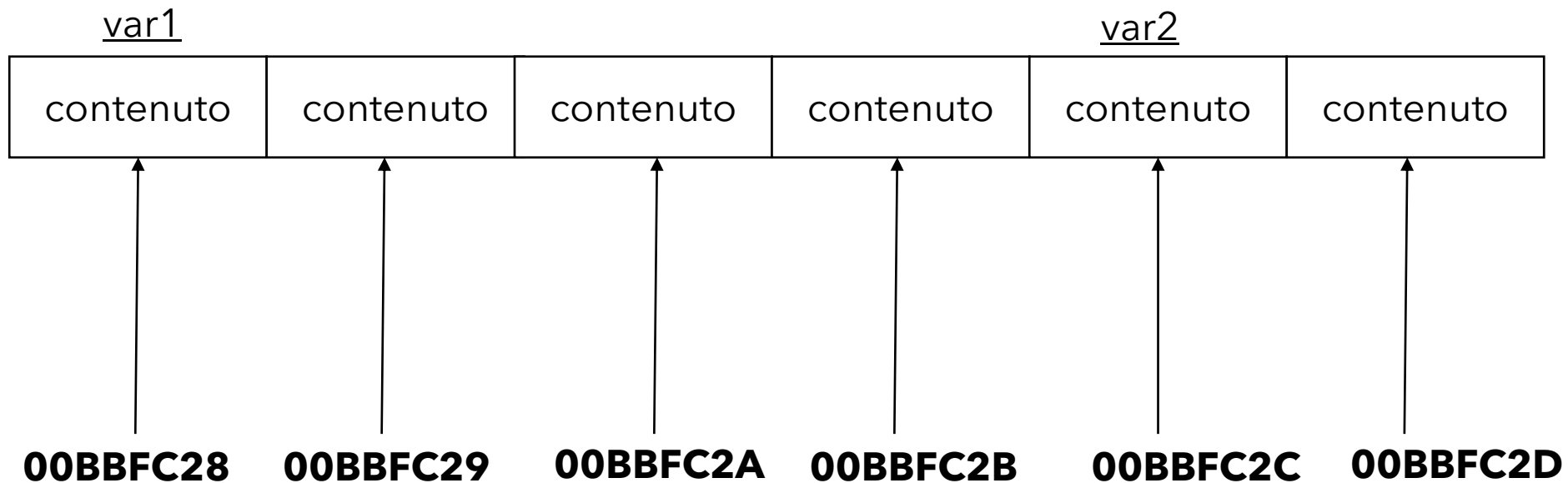
**Nome di variabile: l-value
(left value)**

**costante: r-value (right
value)**

**Un nome di variabile può essere utilizzato
come r-value? Una costante può essere
utilizzata come l-value?**

Indirizzi di memoria (memory addresses)

- La memoria di un computer è una sequenza di byte, ognuno dei quali è dotato di un proprio indirizzo



Indirizzi di memoria (memory addresses)

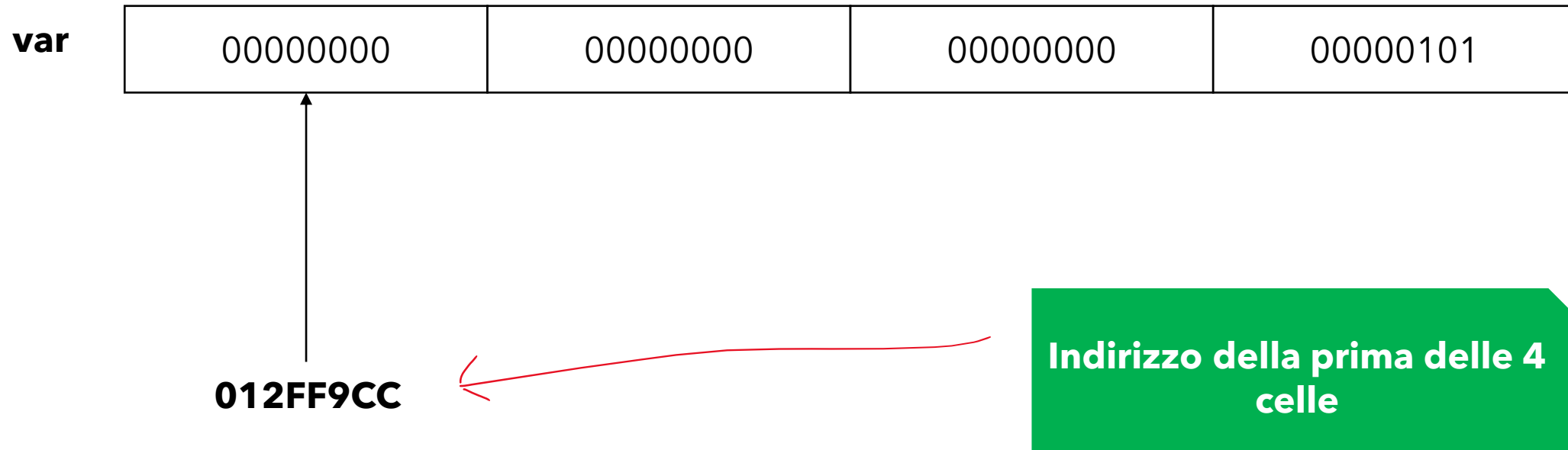
```
int var = 5;  
cout << "var's content is " << var << endl;  
cout << "var occupies " << sizeof(var) << " bytes in memory" << endl;  
cout << "var's address is " << &var << endl;  
cout << "var's address' occupies " << sizeof(&var) << " bytes in memory, or "  
    << 8*sizeof(&var) << " bits" << endl;
```

operatore unario **address of**

```
var's content is 5  
var occupies 4 bytes in memory  
var's address is 00CFFEB0  
var's address' occupies 4 bytes in memory, or 32 bits
```

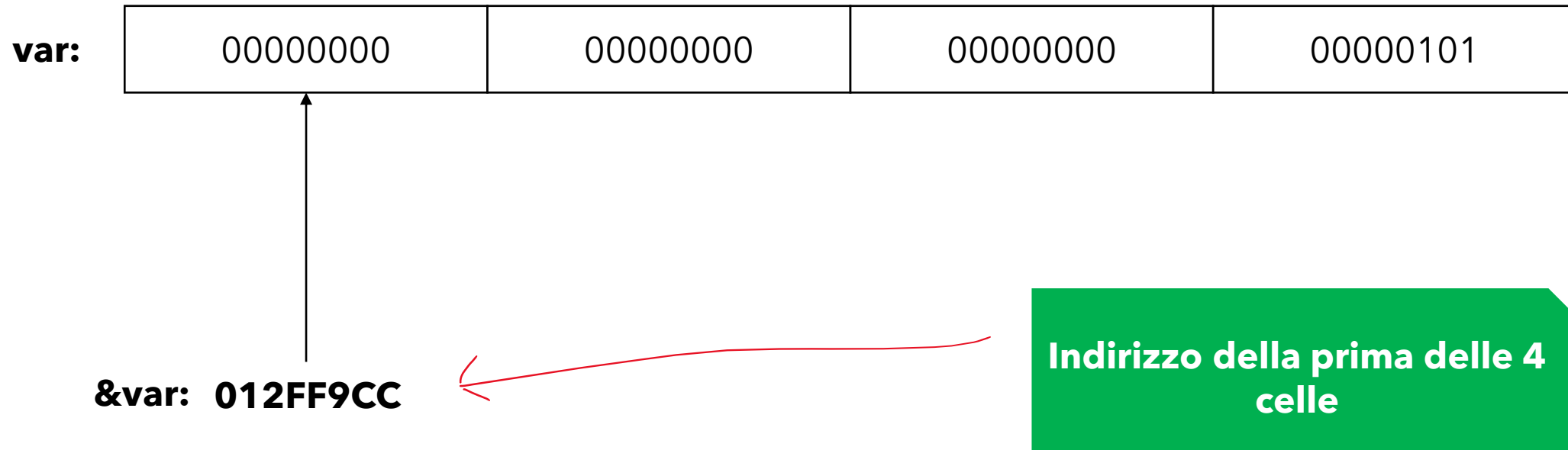
Indirizzi di memoria (memory addresses)

La variabile **int** var occupa 4 byte in memoria (32 bit)



Indirizzi di memoria (memory addresses)

La variabile **int** var occupa 4 byte in memoria (32 bit)



Indirizzi di memoria (memory addresses)

`&(5);` costante intera (**integer literal**)



Secondo voi si può fare? Compila?

Puntatori (pointers)

- Possiamo memorizzare e manipolare gli indirizzi di memoria
- Una variabile contenente un indirizzo di memoria si chiama **puntatore**
- I puntatori in C++ sono tipizzati (come tutte le variabili)
- Dare un tipo ad un puntatore significa specificare il tipo dell'oggetto puntato
- Se un puntatore non avesse un tipo il compilatore non saprebbe quante celle di memoria costituiscono l'oggetto puntato

Puntatori (pointers)

```
int var = 5;  
int* p = &var;
```

- **var** è una variabile di tipo int
- **p** è una variabile di tipo int* (puntatore ad **int**)
- alla variabile **p** viene assegnato l'indirizzo di **var** tramite l'operatore unario **address of** (&)

Puntatori (pointers)

```
int var = 5;  
int* p = &var;
```

address: 012FF9CC

var:

| | | | |
|----------|----------|----------|----------|
| 00000000 | 00000000 | 00000000 | 00000101 |
|----------|----------|----------|----------|

p:

| |
|----------|
| 012FF9CC |
|----------|

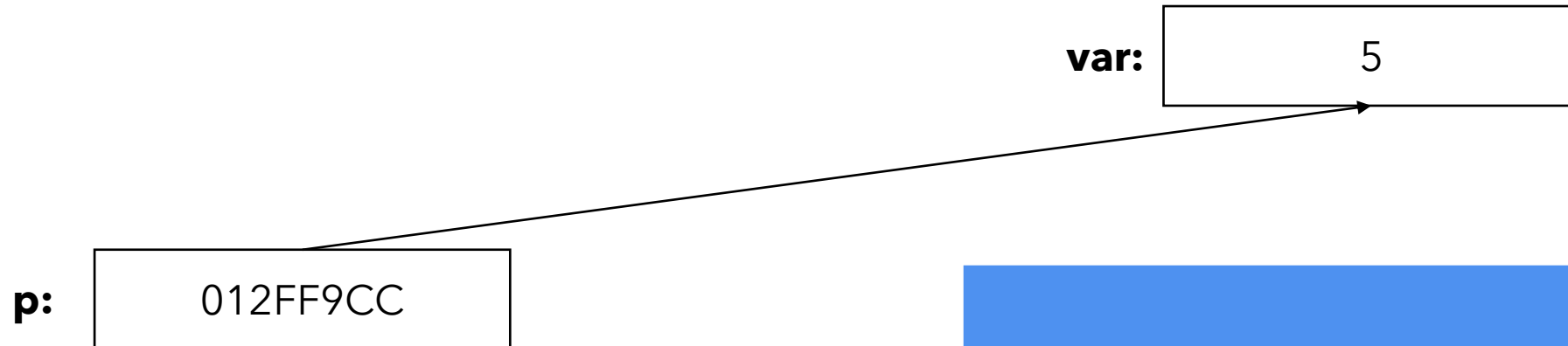


Sarebbero 4 celle dato che la dimensione di un indirizzo è 4 byte... ma per non appesantire la trattazione ne ho disegnata solo una

Puntatori (pointers)

```
int var = 5;  
int* p = &var;
```

Una rappresentazione più sintetica



In gergo informatico: **p** punta a **var**

Puntatori (pointers)

```
char c = 'a';  
char* p = &c;
```

Spiegare il significato di queste istruzioni e fornire una rappresentazione grafica dello stato della memoria

Puntatori (pointers)

- dato un puntatore **p** si vuole accedere all'oggetto puntato
- si utilizza l'operatore unario ***** (**dereferenziazione**)

```
char c = 'a';  
char* c_ptr = &c;  
cout << *c_ptr << endl;
```

Cosa stampa?

Puntatori (pointers)

```
int i = 15;  
int* i_ptr = &i;  
int j = i_ptr;
```

**Si può fare? Al compilatore piacerà?
In effetti il valore di un puntatore è
semplicemente un numero... perché non
assegnarlo ad una variabile int?**

Puntatori (pointers)

```
int x = 10;  
int* p = &x;  
*p = 7;  
int x2 = *p;  
int* p2 = &x2;  
p2 = p;  
p = &x2;
```

Il puntatore nullo (null pointer)

```
int* p = nullptr;  
*p = *p + 2;
```

Dereferenziazione di un puntatore nullo. Boooooom

- **p** è una variabile di tipo `int*` (puntatore a `int`) a cui viene assegnato il valore costante **`nullptr`**
- se un puntatore ha valore **`nullptr`** non punta ad alcun oggetto
- dereferenziare un puntatore nullo provoca un errore a runtime. Il programma «crasha»
- quindi le righe di codice mostrate sopra vengono compilate correttamente, ma il programma *crasherà* quando verrà eseguita la prima istruzione di dereferenziazione, che è **`*p`**

Cosa sono questi array?

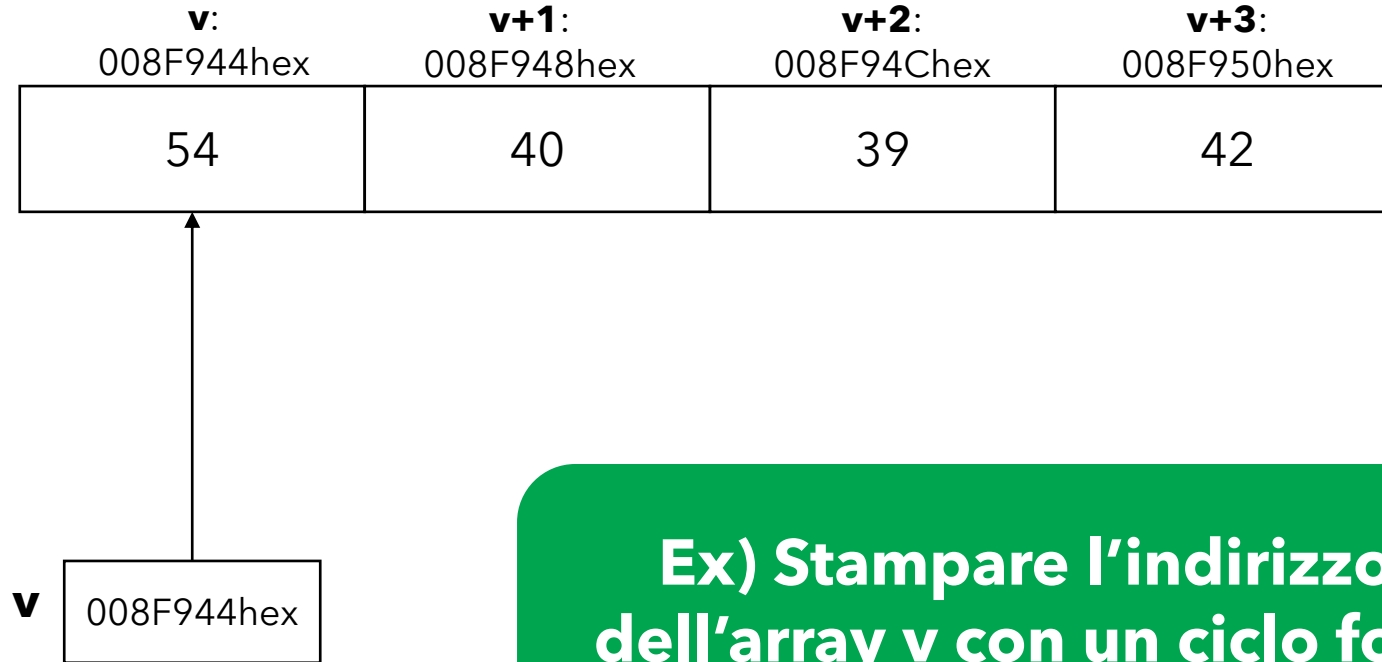
```
int v[] = {54, 40, 39, 42};  
cout << "value of v is: " << v << '\n' << "address of v's first item is: " << &v[0]  
<< '\n';  
int* p0 = v;  
cout << "value of p0 is: " << p0 << '\n';
```

```
value of v is:          008FF944  
address of v's first item is: 008FF944  
value of p0 is:         008FF944
```

Il nome di un array non è altro che un puntatore al suo primo elemento!

Cosa sono questi array? Aritmetica dei puntatori

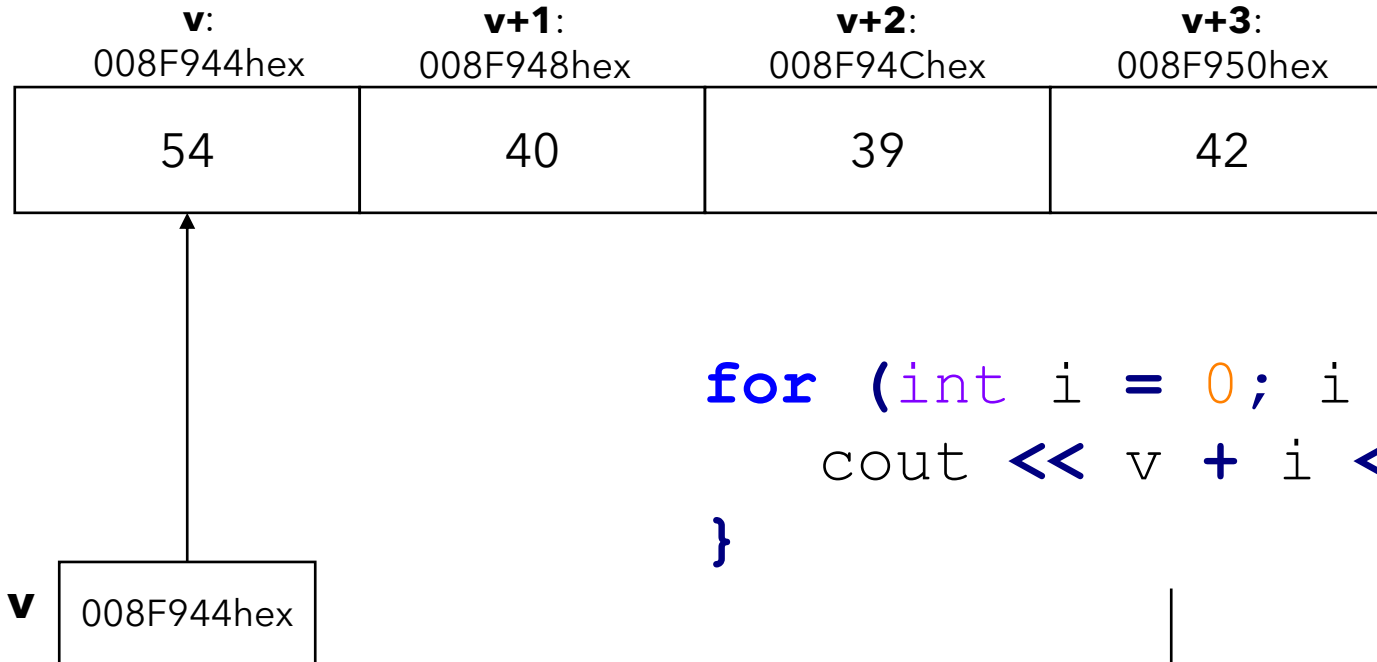
```
int v[] = {54, 40, 39, 42};
```



Ex) Stampare l'indirizzo di ciascun elemento dell'array v con un ciclo for. Separare gli indirizzi con un carattere di tabulazione orizzontale

Cosa sono questi array? Aritmetica dei puntatori

```
int v[] = {54, 40, 39, 42};
```

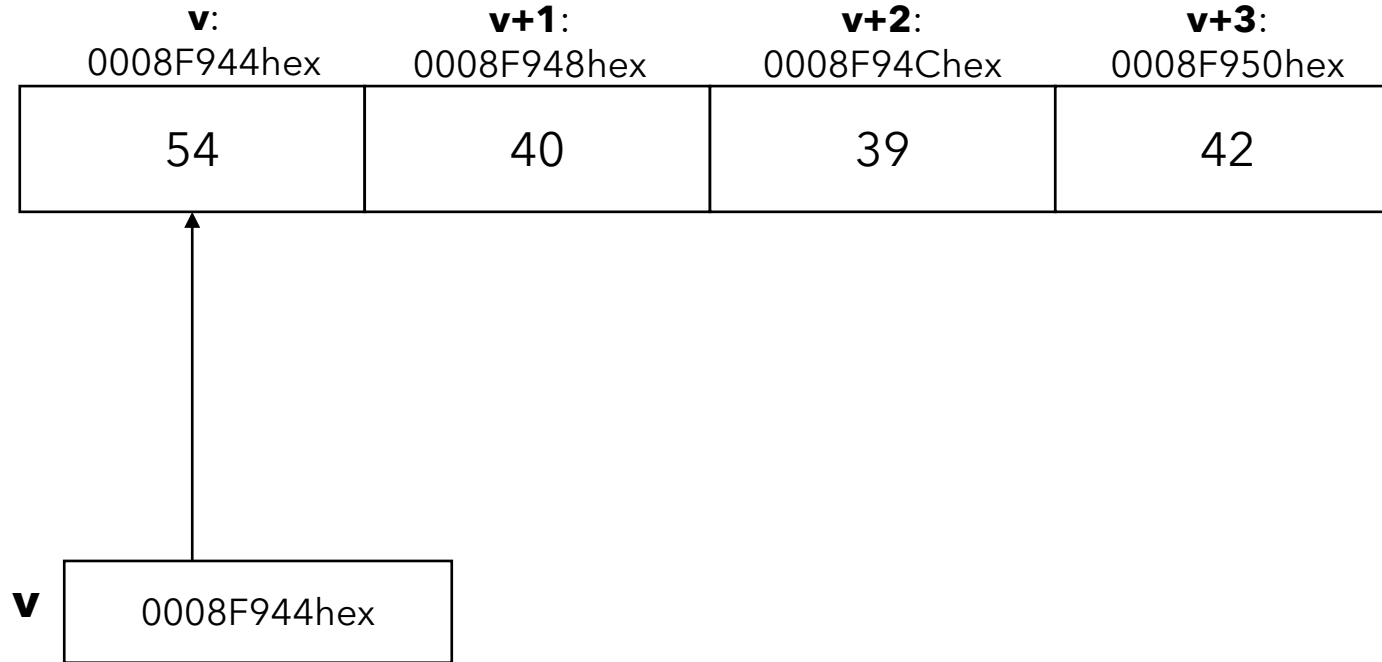


```
for (int i = 0; i < 4; i++) {  
    cout << v + i << '\t';  
}
```

Scrivere il programma che fa la stessa cosa, ma utilizzando l'operatore & per ottenere gli indirizzi

Cosa sono questi array? Aritmetica dei puntatori

```
int v[] = {54, 40, 39, 42};
```



Scrivere un programma che stampa tutti gli elementi dell'array dal primo all'ultimo, ma utilizzando il fatto che $v + i$ è l'indirizzo dell' i -esimo elemento

Riferimenti

- I puntatori sono scomodi perché:
 - Richiedono una sintassi scomoda
 - Bisogna stare attenti a non dereferenziare puntatori nulli
- **Un riferimento è un alias di una variabile. L'accesso al contenuto del riferimento ha una sintassi che non differisce da quella per l'accesso al contenuto di una variabile non puntatore**

Riferimenti

```
int y = 10;  
int& r = y;
```

```
cout << "y's content is: " << y << endl;  
cout << "r's content is: " << r << endl;  
cout << "y's address is: " << &y << endl;  
cout << "r's address is: " << &r << endl;
```


```
r = 20;  
cout << "y's content is: " << y << endl;  
cout << "r's content is: " << r << endl;  
cout << "y's address is: " << &y << endl;  
cout << "r's address is: " << &r << endl;
```

**r è un alias di y.
Una modifica al valore di r
comporta la stessa modifica
al valore di y**

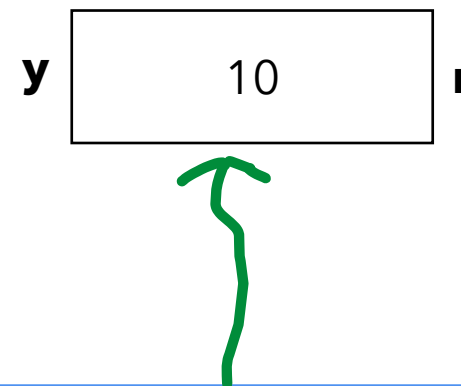
```
y's content is: 10  
r's content is: 10  
y's address is: 0073FA48  
r's address is: 0073FA48  
y's content is: 20  
r's content is: 20  
y's address is: 0073FA48  
r's address is: 0073FA48
```

Riferimenti

```
int y = 10;  
int &r = y;
```



ATTENZIONE: questo & non c'entra un bel niente con l'«address-of»



In pratica abbiamo una cella di memoria con 2 nomi!
Si può fare una cosa simile con i puntatori, ma per accedere al contenuto di r utilizzerò la sintassi che utilizzo per accedere al contenuto di y

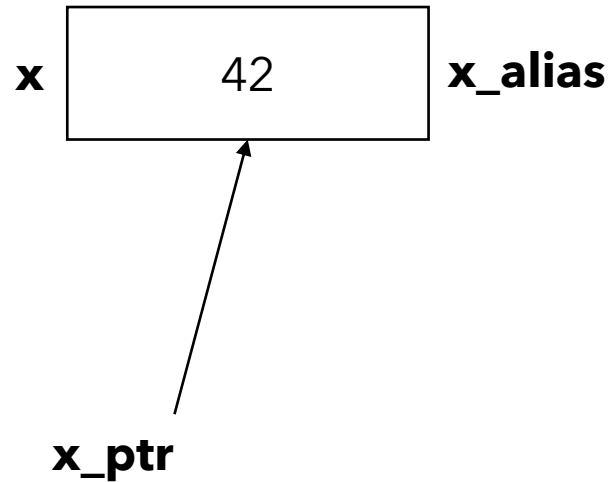
Riferimenti

```
int x = 42;
```

```
int* x_ptr = &x;  
int& x_alias = x;
```

```
cout << "access to the content of x through the pointer x_ptr:
```

```
cout << "access to the content of x through reference (alias) x_alias: "
```



Sintassi «antipatica» ☹️

```
" << *x_ptr << '\n';  
" << x_alias;
```

Sintassi «simpatica» 😊

Riferimenti

```
int y = 10;  
int& y_alias = y;  
y = 7;  
cout << y_alias << endl;  
y_alias = 8;  
cout << y << endl;  
cout << &y_alias << endl;  
cout << &y << endl;  
cout << *y;  
cout << *y_alias;
```

Riferimenti

```
int y = 10;
int& y_alias = y;
y = 7;
cout << y_alias << endl; //prints 7
y_alias = 8;
cout << y << endl; //prints 8
cout << &y_alias << endl; //prints address of y_alias
cout << &y << endl; //prints address of y
cout << *y; //cannot dereference y, it's not a pointer
cout << *y_alias; //cannot dereference y_alias, it's not a pointer
```

Cosa sono questi array a 2 dimensioni?

```
int mat[][3] = {{6, 5, 3}, {4, 9, 8}, {9, 1, 3}};
```

| RAM | | | | | |
|-----|---|---|---|--|--|
| | | | | | |
| | | | | | |
| | 6 | 5 | 3 | | |
| | 4 | 9 | 8 | | |
| | 9 | 1 | 3 | | |
| | | | | | |
| | | | | | |



mat è allocato così in memoria? NO!!!

Cosa sono questi array a 2 dimensioni?

```
int mat[][3] = {{6, 5, 3}, {4, 9, 8}, {9, 1, 3}};
```

```
cout << mat << endl; //mat is the pointer to the matrix' first element  
cout << mat + 1 << endl; //mat is the pointer to the matrix' second element  
cout << mat + 2 << endl; //mat is the pointer to the matrix' third element
```

0057F8E4

0057F8F0

0057F8FC

0057F8F0 - 0057F8E4 = 12 (decimale)

0057F8FC - 0057F8F0 = 12 (decimale)

**Gli elementi di mat
occupano 12 byte.
Spiegare perché.**

Cosa sono questi array a 2 dimensioni?

```
int mat[][3] = {{6, 5, 3}, {4, 9, 8}, {9, 1, 3}};
```

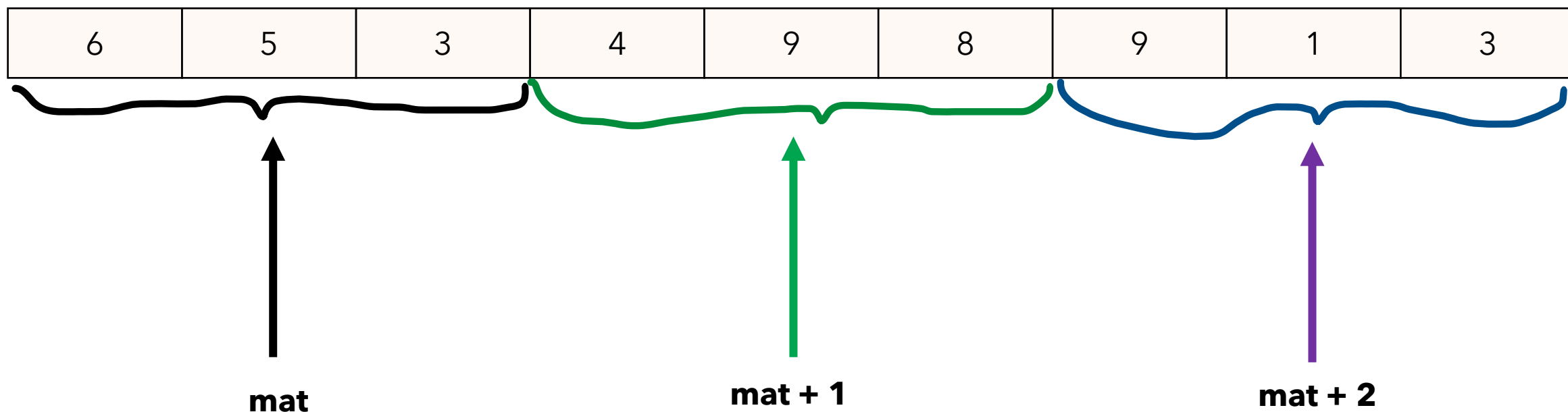
| RAM | | | | | |
|-----|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | 6 | 5 | 3 | 4 | 9 |
| 8 | 9 | 1 | 3 | | |
| | | | | | |
| | | | | | |
| | | | | | |



È allocato così!

Cosa sono questi array a 2 dimensioni?

```
int mat[][3] = {{6, 5, 3}, {4, 9, 8}, {9, 1, 3}};
```



mat è un puntatore ad un array, quindi un puntatore ad un puntatore, ma fermiamoci qui per ora

Puntatore a costante

```
void f(const int* p) {  
    if (p) {  
        cout << *p;  
        *p = *p + 1;  
    }  
}
```

error C3892: 'p': impossibile assegnare a una variabile const

`const int*` p significa: p punta ad un intero costante, ossia, l'intero a cui punta p non può essere modificato

Concetto diverso: puntatore costante

```
int v[10] = {1, 5, 4, 5, 8, 6, 5, 4, 2, 1};  
int* const p = v;  
*p = 99;    //gets compiled  
p++;        //doesn't get compiled
```

Osservazioni? Farsi aiutare dal titolo della slide