

Puntatori e riferimenti **(*pointers and references*)**

Liceo G.B. Brocchi - Bassano del Grappa (VI)
Liceo Scientifico - opzione scienze applicate
Giovanni Mazzocchin

L-values, R-values

```
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

| <u>var1</u> | | | | <u>var2</u> | |
|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| contenuto | contenuto | contenuto | contenuto | contenuto | contenuto |
| 00BBFC28 | 00BBFC29 | 00BBFC2A | 00BBFC2B | 00BBFC2C | 00BBFC2D |

Indirizzi di memoria

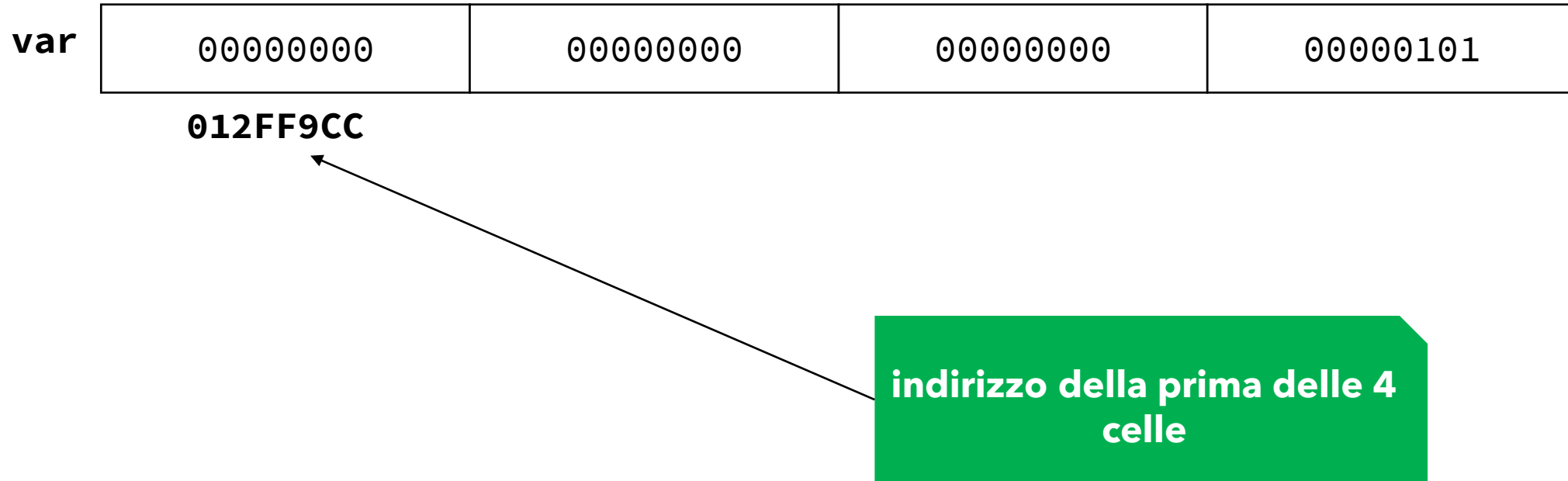
&: operatore unario **address of**

```
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;
```

```
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

La variabile `int var` potrebbe occupare 4 byte in memoria (32 bit)
Su molti sistemi effettivamente gli `int` occupano 4 byte



Indirizzi di memoria

| | | | | |
|------------|-----------------|----------|----------|----------|
| var | 00000000 | 00000000 | 00000000 | 00000101 |
| | 012FF9CC | | | |

&var: 012FF9CC

Indirizzi di memoria

`&(5);`

secondo voi si può fare? Compila?

Puntatori

- 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

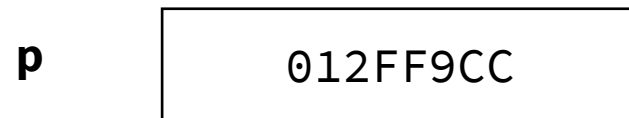
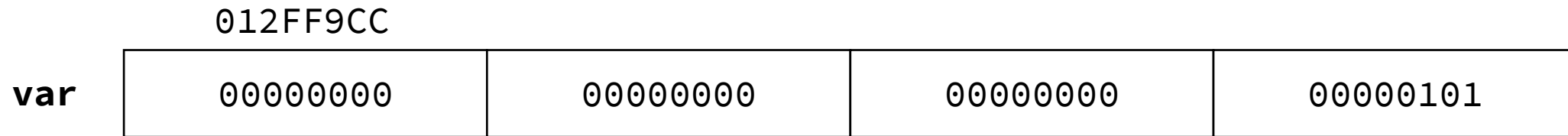
```
int var = 5;  
int* p = &var;
```

- **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** ottenuto con l'operatore unario **address of** (&)

Puntatori

NB: di fatto, su molti sistemi, i byte sono ordinati al contrario

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

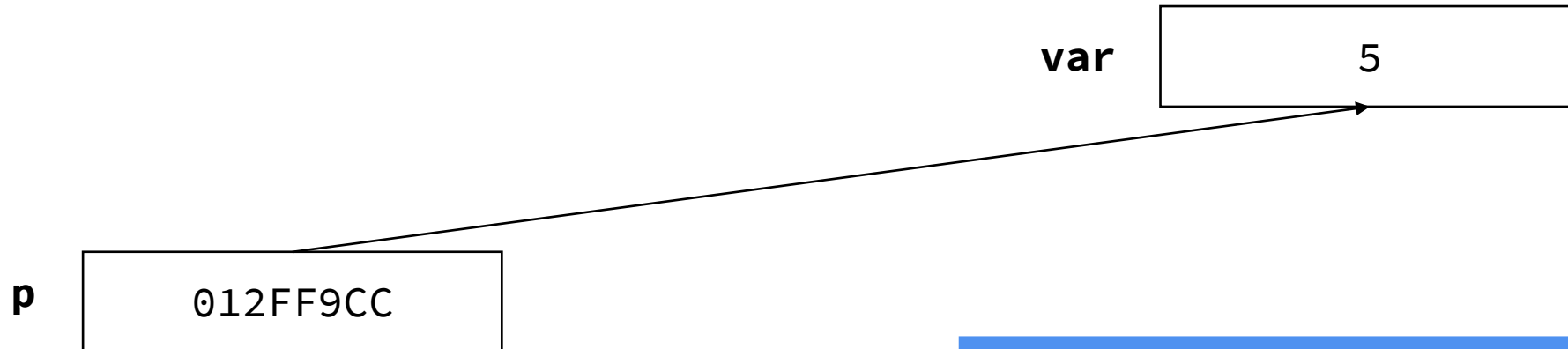


sarebbero 4 celle anche per l'indirizzo, dato che la dimensione di un indirizzo è 4 byte... ma per non appesantire la trattazione ne mostriamo solo una

Puntatori

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

una rappresentazione più sintetica



in gergo informatico: *p punta a var*

Puntatori

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

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

Puntatori

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

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

cosa stampa?

Puntatori

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

**mostrare lo stato della
memoria al termine di
ciascuna istruzione**

Puntatori

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

dereferenziazione di un puntatore nullo: *undefined behaviour*

- **p** è una variabile di tipo `int*` (puntatore a `int`) a cui viene assegnato il valore **`nullptr`**
- se un puntatore ha valore **`nullptr`**, allora non punta ad alcuna area di memoria
- dereferenziare un puntatore nullo può provocare un errore a runtime. Il programma potrebbe crashare
- quindi, le righe di codice mostrate sopra vengono compilate correttamente, ma il programma probabilmente crasherà quando verrà eseguita la prima istruzione di dereferenziazione, che è **`*p`**

Array come puntatori

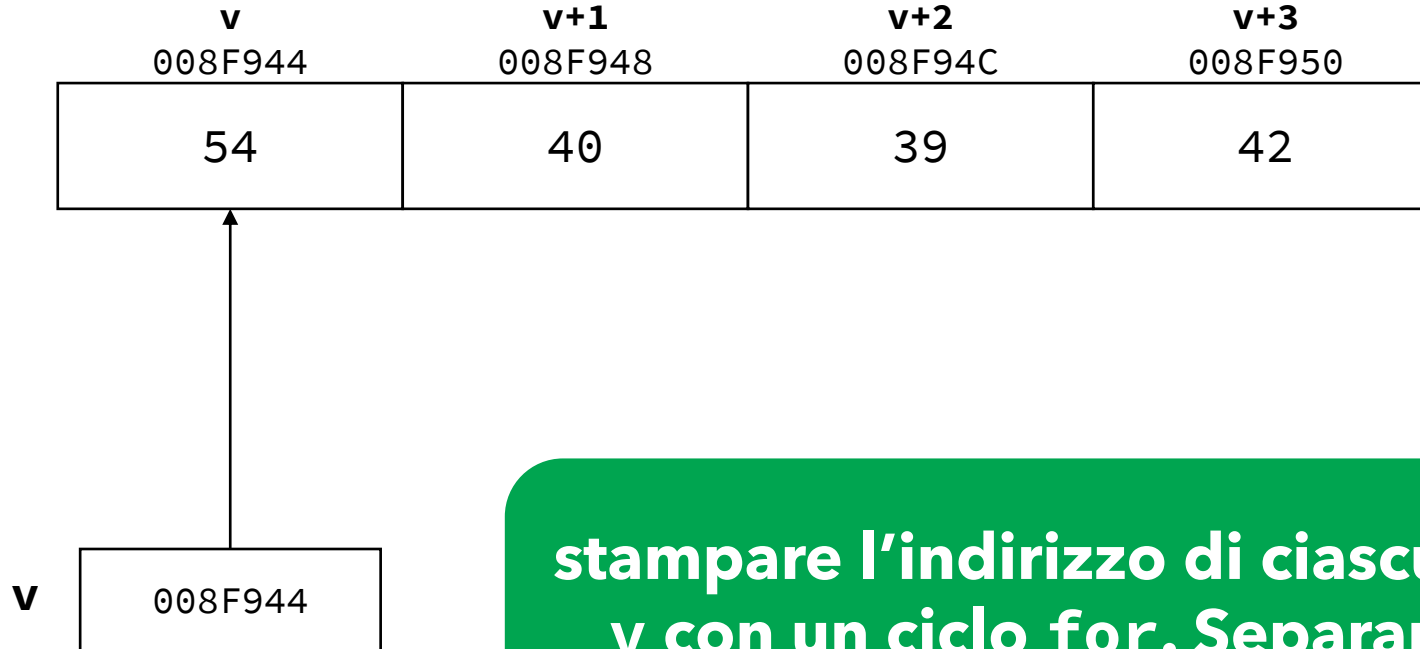
```
int v[] = {54, 40, 39, 42};  
cout << "value of v is: " << v << '\n' << "address of v's first item is: " << &v[0]  
<< '\n';  
int* p0 = v;  
cout << "value of p0 is: " << p0 << '\n';
```

```
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!

Array come puntatori

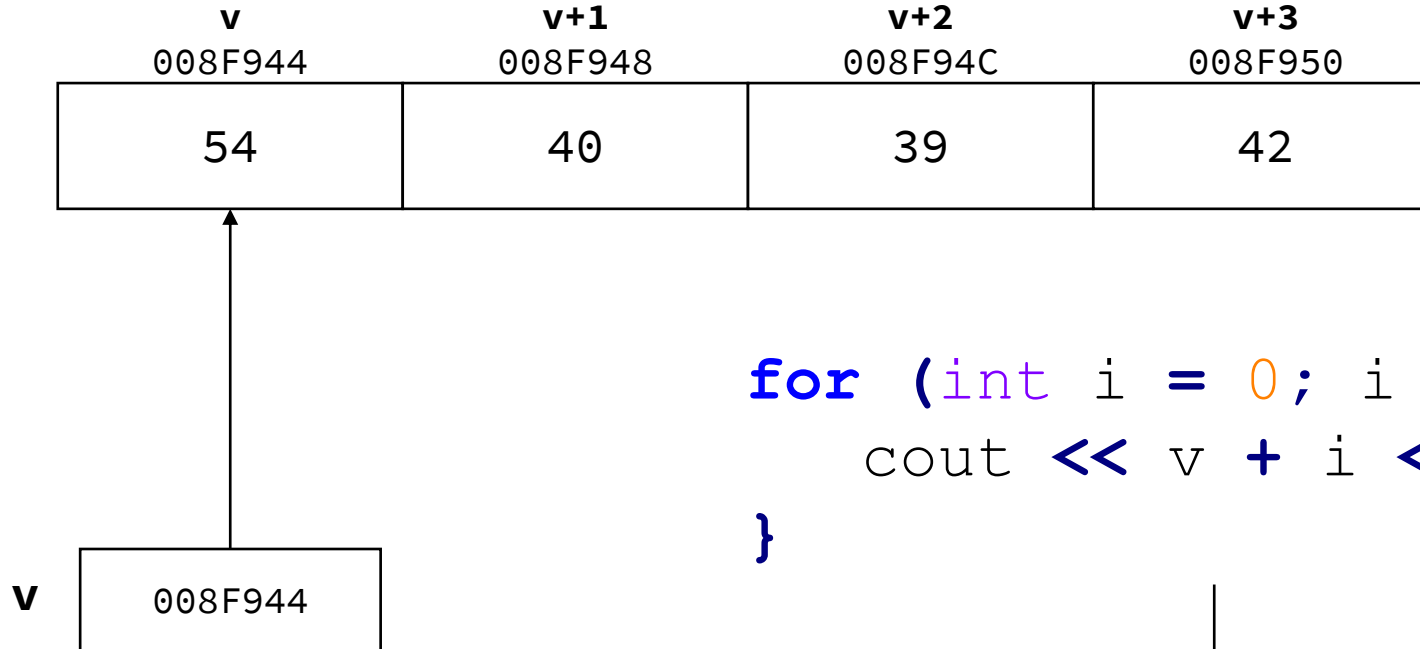
```
int v[] = {54, 40, 39, 42};
```



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

Aritmetica dei puntatori

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

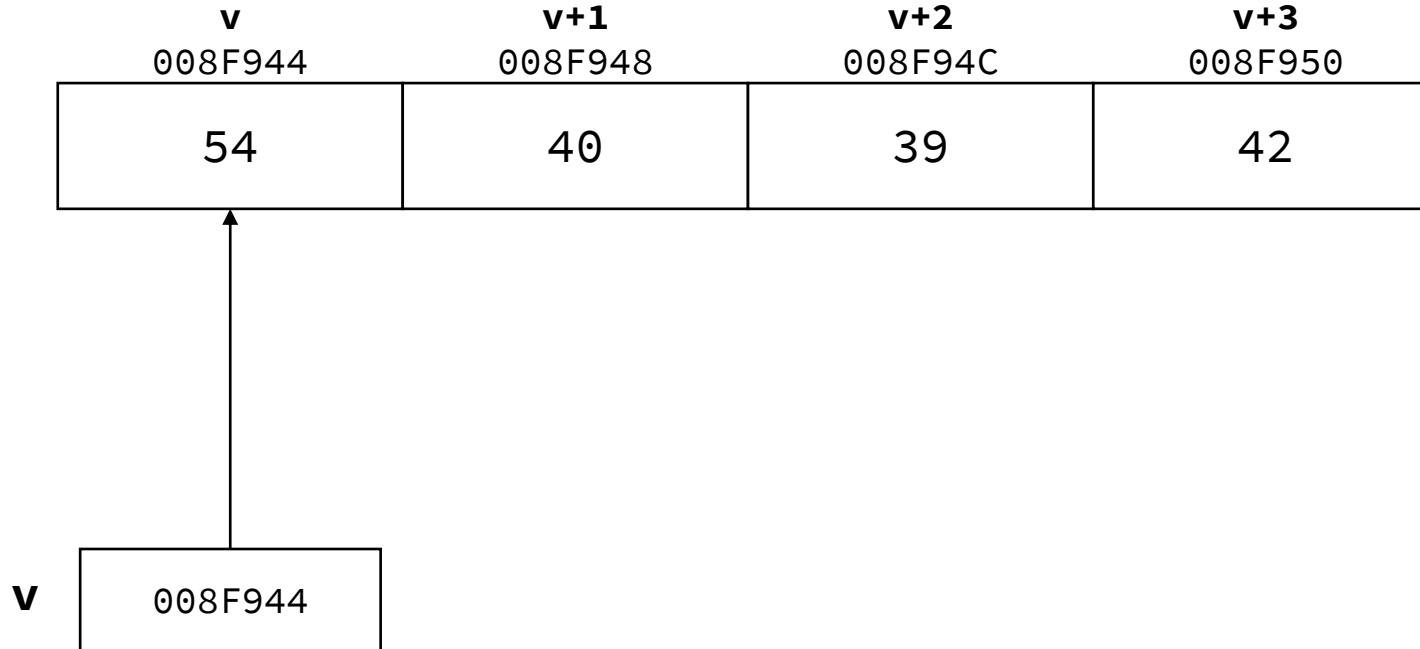


```
for (int i = 0; i < 4; i++) {  
    cout << v + i << '\t';  
}
```

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

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 i puntatori nulli
- Un **riferimento** è un alias di una variabile. L'accesso al contenuto di un 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;
```

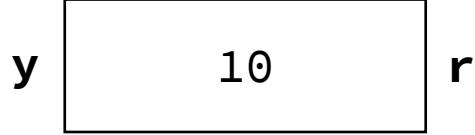
```
r = 20;  
cout << "y's content is: " << y << endl;  
cout << "r's content is: " << r << endl;  
cout << "y's address is: " << &y << endl;  
cout << "r's address is: " << &r << endl;
```

**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 niente con l'& address of

in pratica abbiamo un'area di memoria
con 2 nomi!
Per accedere al contenuto di r utilizzerò la
sintassi che utilizzo per accedere al
contenuto di y

Riferimenti

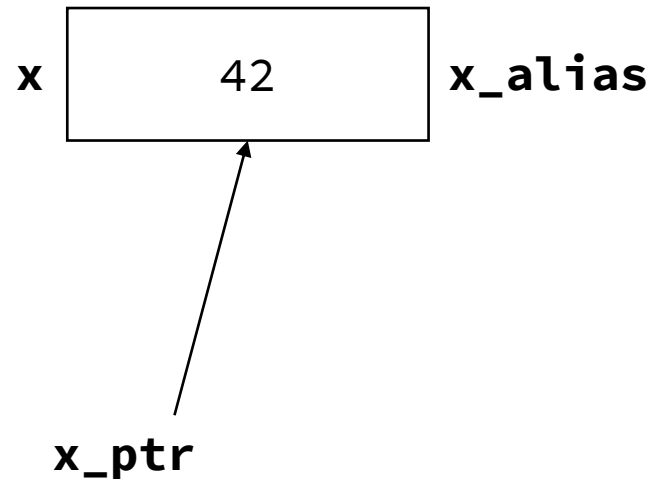
```
int x = 42;
```

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

```
int& x_alias = x;
```

```
cout << "access to the content of x through the pointer x_ptr: " << *x_ptr << '\n';
```

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



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;  
y_alias = 8;  
cout << y << endl;  
cout << &y_alias << endl;  
cout << &y << endl;  
cout << *y;  
cout << *y_alias;
```

Array a 2 dimensioni come puntatori

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

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

**mat è allocato così in memoria? NO!
Come è allocato?**

Array a 2 dimensioni come puntatori

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

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

0057F8E4

0057F8F0

0057F8FC

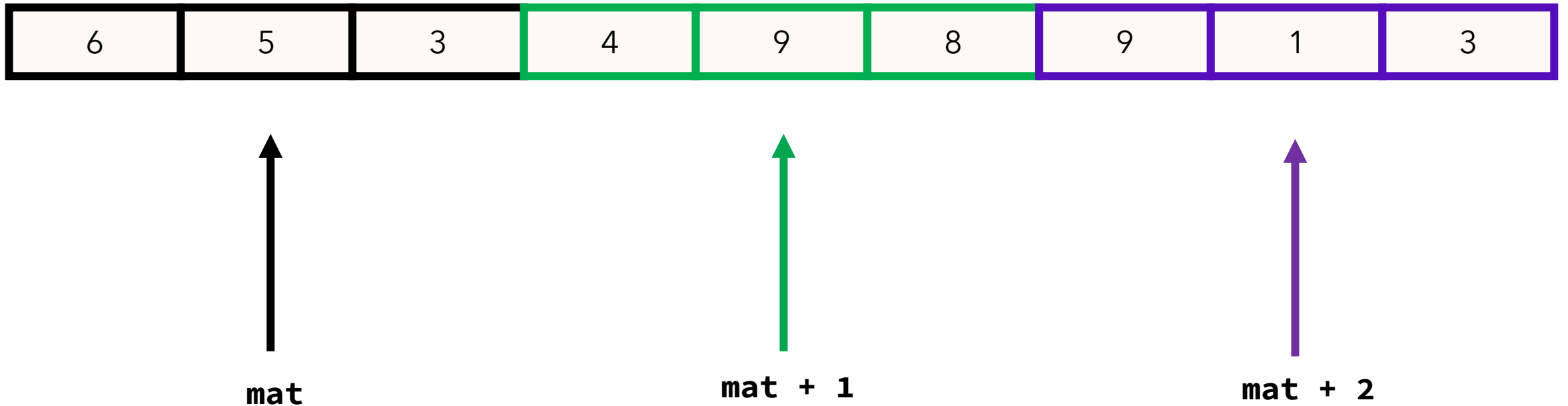
**gli elementi di mat
occupano 12 byte,
perché?**

0057F8F0 - 0057F8E4 = 12 (decimale)

0057F8FC - 0057F8F0 = 12 (decimale)

Array a 2 dimensioni come puntatori

```
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

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: p punta ad un intero che non può essere modificato dereferenziando p

Puntatore costante

```
int v[10] = {1, 5, 4, 5, 8, 6, 5, 4, 2, 1};  
int* const p = v;  
*p = 99;    //ok  
p++;        //error
```

osservazioni? Farsi aiutare dal titolo della slide

Da vedere a casa

- [Essentials: Hidden Pointers – Computerphile](#)
- [you will never ask about pointers again after watching this video](#)
- [you will never ask about pointer arithmetic after watching this video](#)