

Fondamenti di Programmazione (A)

I3 - Strutture dati (Stringhe e struct)

Vincenzo Arceri - Università degli Studi di Parma - vincenzo.arceri@unipr.it

Puntate precedenti

- Array mono-dimensionali
 - Array statici
 - Array semi-dinamici
- Array bi-dimensionali
 - Matrici

Stringhe

- Una stringa è una sequenza di $n \geq 0$ caratteri
- Solitamente per identificare una stringa, viene delimitata da doppi (C++) o singoli apici (Python ma non C++)

`"hello"`

`"hello world"`

`" "`

stringa vuota

Stringhe

- Una stringa è una sequenza di $n \geq 0$ caratteri
 - Solitamente per identificare una stringa, viene delimitata da doppi (C++) o singoli apici (Python ma non C++)

`"hello"`

`"hello world"`

`" "`

stringa vuota

- Alcune operazioni su stringhe: lunghezza, estrarre il carattere in una certa posizione, estrarre la sottostringa fra due posizioni, concatenazione di due stringhe...

Stringhe

Implementazione tramite array (C-style)

```
"hello"
```

Stringhe

Implementazione tramite array (C-style)

"hello"

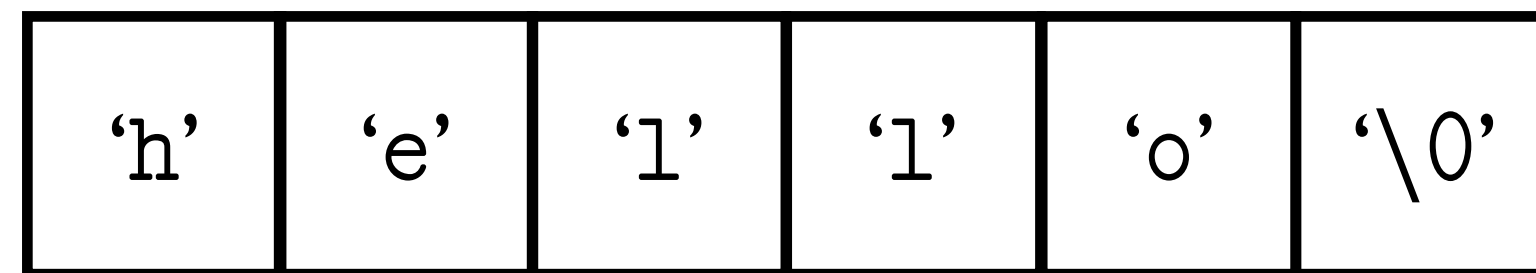
- In C, una stringa è un **array di caratteri** che termina con il carattere di terminazione `'\0'`

Stringhe

Implementazione tramite array (C-style)

"hello"

- In C, una stringa è un **array di caratteri** che termina con il carattere di terminazione `'\0'`

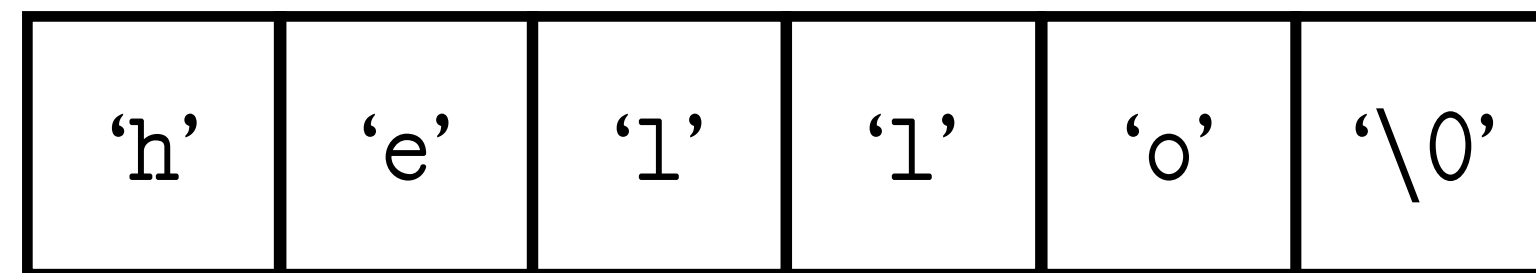


Stringhe

Implementazione tramite array (C-style)

"hello"

- In C, una stringa è un **array di caratteri** che termina con il carattere di terminazione `'\0'`



Questa è una stringa ("hello")

Stringhe

Implementazione tramite array (C-style)

"hello"

- In C, una stringa è un **array di caratteri** che termina con il carattere di terminazione `'\0'`

'h'	'e'	'l'	'l'	'o'	'\0'
-----	-----	-----	-----	-----	------

'h'	'e'	'l'	'l'	'o'
-----	-----	-----	-----	-----

Questa è una stringa ("hello")

Stringhe

Implementazione tramite array (C-style)

"hello"

- In C, una stringa è un **array di caratteri** che termina con il carattere di terminazione `'\0'`

'h'	'e'	'l'	'l'	'o'	'\0'
-----	-----	-----	-----	-----	------

Questa è una stringa ("hello")

'h'	'e'	'l'	'l'	'o'
-----	-----	-----	-----	-----

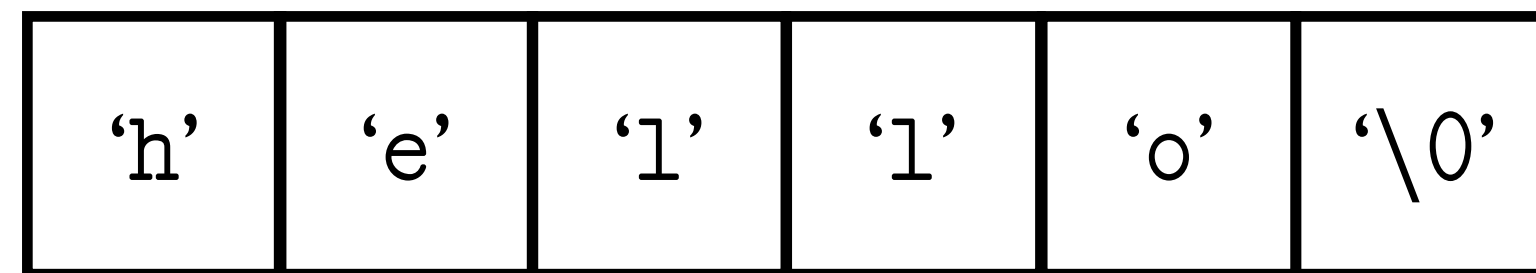
Questa è **non** una stringa, ma
è solo un array di caratteri

Stringhe

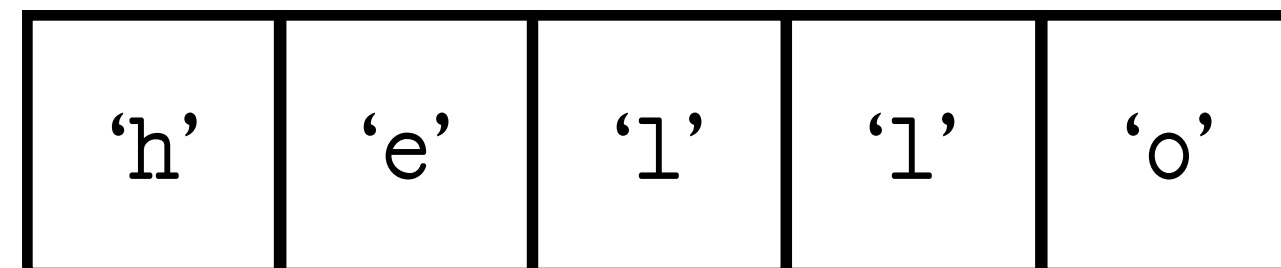
Implementazione tramite array (C-style)

"hello"

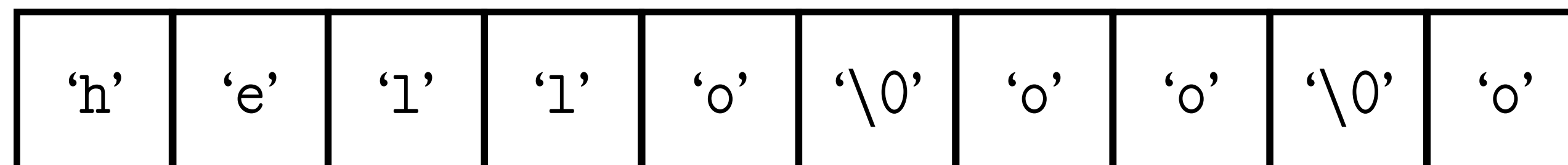
- In C, una stringa è un **array di caratteri** che termina con il carattere di terminazione `'\0'`



Questa è una stringa ("hello")



Questa è **non** una stringa, ma
è solo un array di caratteri



Stringhe

Implementazione tramite array (C-style)

"hello"

- In C, una stringa è un **array di caratteri** che termina con il carattere di terminazione `'\0'`

'h'	'e'	'l'	'l'	'o'	'\0'
-----	-----	-----	-----	-----	------

Questa è una stringa ("hello")

'h'	'e'	'l'	'l'	'o'
-----	-----	-----	-----	-----

Questa è **non** una stringa, ma
è solo un array di caratteri

'h'	'e'	'l'	'l'	'o'	'\0'	'o'	'o'	'\0'	'o'
-----	-----	-----	-----	-----	------	-----	-----	------	-----

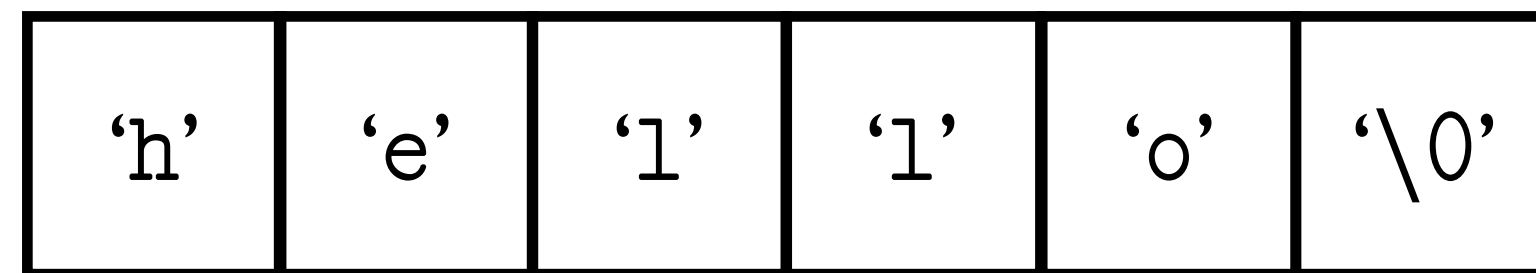
Questa è una stringa ("hello")

Stringhe

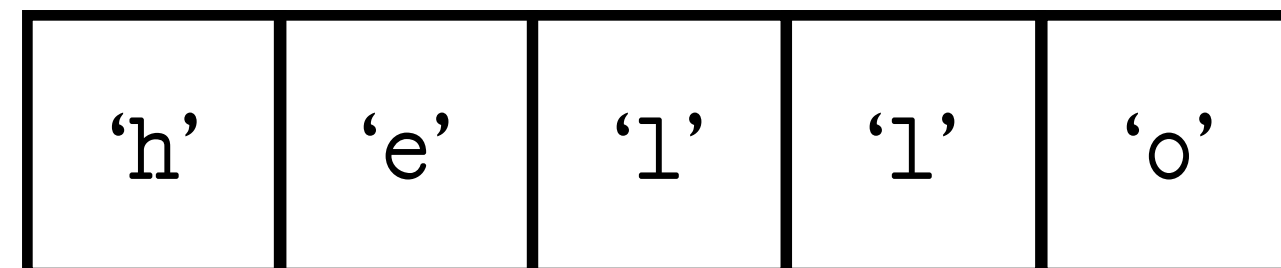
Implementazione tramite array (C-style)

"hello"

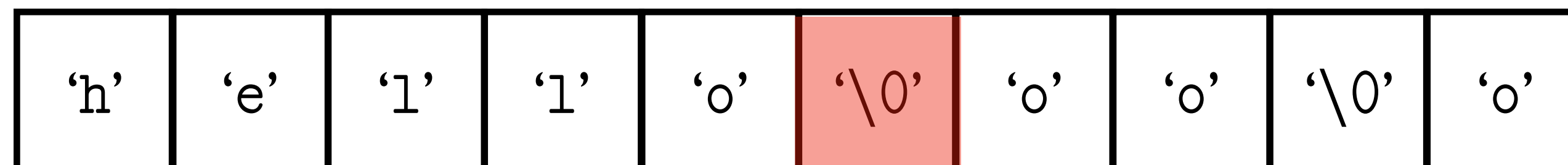
- In C, una stringa è un **array di caratteri** che termina con il carattere di terminazione `'\0'`



Questa è una stringa ("hello")



Questa è **non** una stringa, ma è solo un array di caratteri



Questa è una stringa ("hello")

Stringhe

Implementazione tramite array (C-style)

"hello"

'h'	'e'	'l'	'l'	'o'	'\0'
-----	-----	-----	-----	-----	------

Stringhe

Implementazione tramite array (C-style)

"hello"

- In C, una stringa " $c_0c_1\dots c_n$ " può essere realizzata con un **array di caratteri** di lunghezza $k \geq n + 1$ tale che

'h'	'e'	'l'	'l'	'o'	'\0'
-----	-----	-----	-----	-----	------

Stringhe

Implementazione tramite array (C-style)

"hello"

- In C, una stringa " $c_0c_1\dots c_n$ " può essere realizzata con un **array di caratteri** di lunghezza $k \geq n + 1$ tale che
 - Le prime n celle dell'array (da 0 a $n - 1$) contengono, in ordine i caratteri della stringa

'h'	'e'	'l'	'l'	'o'	'\0'
-----	-----	-----	-----	-----	------

Stringhe

Implementazione tramite array (C-style)

"hello"

- In C, una stringa " $c_0c_1\dots c_n$ " può essere realizzata con un **array di caratteri** di lunghezza $k \geq n + 1$ tale che
 - Le prime n celle dell'array (da 0 a $n - 1$) contengono, in ordine i caratteri della stringa
 - Il carattere in posizione n contiene `'\0'`

'h'	'e'	'l'	'l'	'o'	'\0'
-----	-----	-----	-----	-----	------

Stringhe

Implementazione tramite array (C-style)

"hello"

- In C, una stringa " $c_0c_1\dots c_n$ " può essere realizzata con un **array di caratteri** di lunghezza $k \geq n + 1$ tale che
 - Le prime n celle dell'array (da 0 a $n - 1$) contengono, in ordine i caratteri della stringa
 - Il carattere in posizione n contiene `'\0'`

'h'	'e'	'l'	'l'	'o'	'\0'
-----	-----	-----	-----	-----	------

'h'	'e'	'l'	'l'	'o'	'\0'	'o'	'o'	'\0'	'o'
-----	-----	-----	-----	-----	------	-----	-----	------	-----

Stringhe

Dichiarazione

```
"hello"
```

Stringhe

Dichiarazione

"hello"

```
char str[6] = {'h', 'e', 'l', 'l', 'o', '\0'};
```

Stringhe

Dichiarazione

"hello"

```
char str[6] = {'h', 'e', 'l', 'l', 'o', '\0'};
```

```
char str[10] = {'h', 'e', 'l', 'l', 'o', '\0', 'a', 'a'};
```

Stringhe

Dichiarazione

"hello"

```
char str[6] = {'h', 'e', 'l', 'l', 'o', '\0'};
```

```
char str[10] = {'h', 'e', 'l', 'l', 'o', '\0', 'a', 'a'};
```

```
char str[] = {'h', 'e', 'l', 'l', 'o', '\0'};
```

Stringhe

Dichiarazione

"hello"

```
char str[6] = {'h', 'e', 'l', 'l', 'o', '\0'};
```

```
char str[10] = {'h', 'e', 'l', 'l', 'o', '\0', 'a', 'a'};
```

```
char str[] = {'h', 'e', 'l', 'l', 'o', '\0'};
```

La lunghezza dell'array viene
inferita dal compilatore

Stringhe

Dichiarazione

"hello"

```
char str[6] = {'h', 'e', 'l', 'l', 'o', '\0'};
```

```
char str[10] = {'h', 'e', 'l', 'l', 'o', '\0', 'a', 'a'};
```

```
char str[] = {'h', 'e', 'l', 'l', 'o', '\0'};
```

La lunghezza dell'array viene
inferita dal compilatore

```
char str[] = "hello";
```


Stringhe

Dichiarazione

"hello"

```
char str[6] = {'h', 'e', 'l', 'l', 'o', '\0'};
```

```
char str[10] = {'h', 'e', 'l', 'l', 'o', '\0', 'a', 'a'};
```

```
char str[] = {'h', 'e', 'l', 'l', 'o', '\0'};
```

La lunghezza dell'array viene
inferita dal compilatore

```
char str[] = "hello";
```

La lunghezza dell'array viene inferita dal compilatore e viene automaticamente
messo il carattere di terminazione '\0'

Stringhe

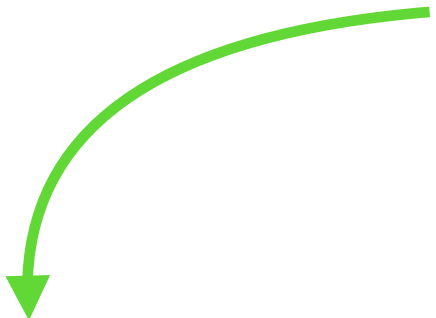
Esercizio

- Data in input una stringa s , calcolare e stampare il numero di vocali presenti in s

Stringhe

Esercizio

- Data in input una stringa *s*, calcolare e stampare il numero di vocali presenti in *s*
- **Problemi con `cin`:**
 - la lettura di una stringa tramite lo stream di input `cin` si interrompe quando si incontra il carattere *spazio* o *a capo*

hello world

`cin >> str;` str conterrà la stringa hello!

- Se la lunghezza della stringa in input è maggiore della dimensione dell'array in cui viene memorizzata la stringa, la stringa in input viene memorizzata lo stesso!

Stringhe

Esercizio

- Data in input una stringa s , calcolare e stampare il numero di vocali presenti in s
- **Problemi con `cin`:**
 - Se la lunghezza della stringa in input è maggiore della dimensione dell'array in cui viene memorizzata la stringa, la stringa in input viene memorizzata lo stesso!

```
char str[5];  
cin >> str;
```

str conterrà la stringa helloworld!

helloworld



La correttezza del programma
è compromessa: stiamo
utilizzando spazio di memoria
che non era riservato/dedicato
per `str`!

Stringhe

`getline` da `iostream`

`s.getline(str, l, d)`

- *s* è uno stream di input (e.g., `cin`)
- *str* è una stringa in `C` (array di caratteri)
- *l* è il numero di caratteri da leggere dallo stream di input *s*
- *d* è un parametro opzionale che indica il carattere delimitatore della stringa
 - se assente, si assume che il suo valore sia `'\n'` (*a capo*)

Stringhe

`getline` da `iostream`

`s.getline(str, l, d)`

- Vengono estratti i caratteri da *s* uno alla volta e memorizzati in *str* finché
 - non si incontra il carattere *d* (che non viene aggiunto a *str*)
 - oppure, non sono stati letti $l - 1$ caratteri

Stringhe

getline da iostream

`s.getline(str, l, d)`

- Vengono estratti i caratteri da *s* uno alla volta e memorizzati in *str* finché
 - non si incontra il carattere *d* (che non viene aggiunto a *str*)
 - oppure, non sono stati letti $l - 1$ caratteri
- Infine, viene memorizzato alla fine di *str* il carattere speciale ‘\0’

```
char str[50];  
cin.getline(str, 50);
```

Leggo da cin al massimo 49 caratteri, fino al raggiungimento del carattere ‘\n’, e li memorizzo in str

Stringhe

Esercizio

- Codifica di una frase: si scriva un programma che legga una frase da tastiera lunga al più 100 caratteri. Il programma deve
 - Stampare la frase letta
 - Costruire una codifica della frase tale che:
 - ogni vocale della frase è seguita dalla lettera 'f' (se la vocale è minuscola) oppure 'F' (se la vocale è maiuscola)

hE11ow0r1d

hEF11ofw0Fr1d

Stringhe

Libreria `cstring`

```
#include <cstring>
```

- La libreria `cstring` contiene una serie di funzioni predefinite per la manipolazione di stringhe

Stringhe

Libreria `cstring`

`strlen(s)`

- Ritorna la lunghezza della stringa `s`

Stringhe

Libreria `cstring`

`strlen(s)`

- Ritorna la lunghezza della stringa `s`

```
char parola[] = "ciao";  
cout << strlen(parola); // stampa 4
```

Stringhe

Libreria `cstring`

`strcpy(s, r)`

- Copia il contenuto della stringa `r` nella stringa `s`

Stringhe

Libreria `cstring`

`strcpy(s, r)`

- Copia il contenuto della stringa `r` nella stringa `s`

```
char s[20];  
char r[] = "helloworld";  
strcpy(s, r);  
cout << s; // Stampa helloworld
```

Stringhe

Libreria `cstring`

`strcat(s, r)`

- Concatena la stringa `r` alla stringa `s`

```
char s[32] = "hello";  
char r[] = " world!";  
strcat(s, r);  
cout << s; // stampa hello world!
```

- Modifica il contenuto della variabile `s`: l'array in cui viene memorizzata la stringa risultante dalla concatenazione deve avere dimensione strettamente maggiore della somma delle dimensioni delle due stringhe concatenate

Stringhe

Libreria `cstring`

`strcmp(s, r)`

- Confronta due stringhe e ritorna
 - 0 se le due stringhe sono uguali
 - < 0 se `s` precede lessicograficamente `r`
 - > 0 se `r` precede lessicograficamente `s`

Stringhe

Libreria cstring

`strcmp(s, r)`

`strcmp("abc", "ab")`

ritorna > 0

`strcmp("casato", "casa")`

ritorna > 0

`strcmp("hello", "hello")`

ritorna 0

`strcmp("abc", "zyx")`

ritorna < 0

Recap

- Tipi primitivi
 - `int`, `bool`, `char`, `float`, ...
- Tipi strutturati
 - `array`
 - mono-dimensionali
 - bi-dimensionali (matrici)
 - stringhe (implementazione C tramite array di caratteri)

Persona?

- Immaginiamo di voler *modellare*, nel nostro programma, una persona
 - Nome
 - Cognome
 - Età

Persona?

- Immaginiamo di voler *modellare*, nel nostro programma, una persona
 - Nome `char nome[32]`
 - Cognome `char cognome[32]`
 - Età `int eta`
- Ma non abbiamo il tipo `persona...`
 - Tre variabili differenti? Sì ma *logicamente* slegate...
 - Array? No!

Persona?

- Immaginiamo di voler *modellare*, nel nostro programma, una persona
 - Nome `char nome[32]`
 - Cognome `char cognome[32]`
 - Età `int eta`
- Ma non abbiamo il tipo `persona`...
 - Tre variabili differenti?
 - Array?

persona me_stesso

nome	“Vincenzo”
cognome	“Arceri”
eta	32

Struct

Dichiarazione di una variabile di tipo struct

- Il tipo struct permette di *aggregare* più variabili di tipo diverso

```
struct {  
     $t_0$   $id_0$  ;  
     $t_1$   $id_1$  ;  
     $t_2$   $id_2$  ;  
    ...  
     $t_n$   $id_n$  ;  
}  $x$  ;
```

Struct

Dichiarazione di una variabile di tipo struct

- Il tipo struct permette di *aggregare* più variabili di tipo diverso

```
struct {  
     $t_0$   $id_0$  ;  
     $t_1$   $id_1$  ;  
     $t_2$   $id_2$  ;  
    ...  
     $t_n$   $id_n$  ;  
}  $x$  ;
```

Campi del tipo struct tale che:


- $\forall i \in [0, n], t_i$ sono tipi qualsiasi
- $\forall i \in [0, n], id_i$ sono identificatori

Struct

Dichiarazione di una variabile di tipo struct

- Il tipo struct permette di *aggregare* più variabili di tipo diverso

```
struct {  
   $t_0$   $id_0$ ;  
   $t_1$   $id_1$ ;  
   $t_2$   $id_2$ ;  
  ...  
   $t_n$   $id_n$ ;  
}  $x$ ;
```



Campo di nome id_0 e di tipo t_0


Struct

Dichiarazione di una variabile di tipo struct

- Il tipo struct permette di *aggregare* più variabili di tipo diverso

```
struct {  
     $t_0$   $id_0$  ;  
     $t_1$   $id_1$  ;  
     $t_2$   $id_2$  ;  
    ...  
     $t_n$   $id_n$  ;  
}  $x$  ;
```

Identificatore (della variabile che sto dichiarando)



Struct

Dichiarazione di una variabile di tipo struct

- Il tipo struct permette di *aggregare* più variabili di tipo diverso

```
struct {  
     $t_0$   $id_0$  ;  
     $t_1$   $id_1$  ;  
     $t_2$   $id_2$  ;  
    ...  
     $t_n$   $id_n$  ;  
}  $x$  ;
```

Identificatore (della variabile
che sto dichiarando)

La variabile x è del tipo
struct costituito da $n + 1$
campi, ciascuno dei quali
chiamato id_0, \dots, id_n e di
tipo t_0, \dots, t_n ,
rispettivamente

Struct

Dichiarazione di una variabile di tipo struct

```
struct {  
    char nome[32];  
    char cognome[32];  
    int eta;  
} me_stesso;
```

me_stesso

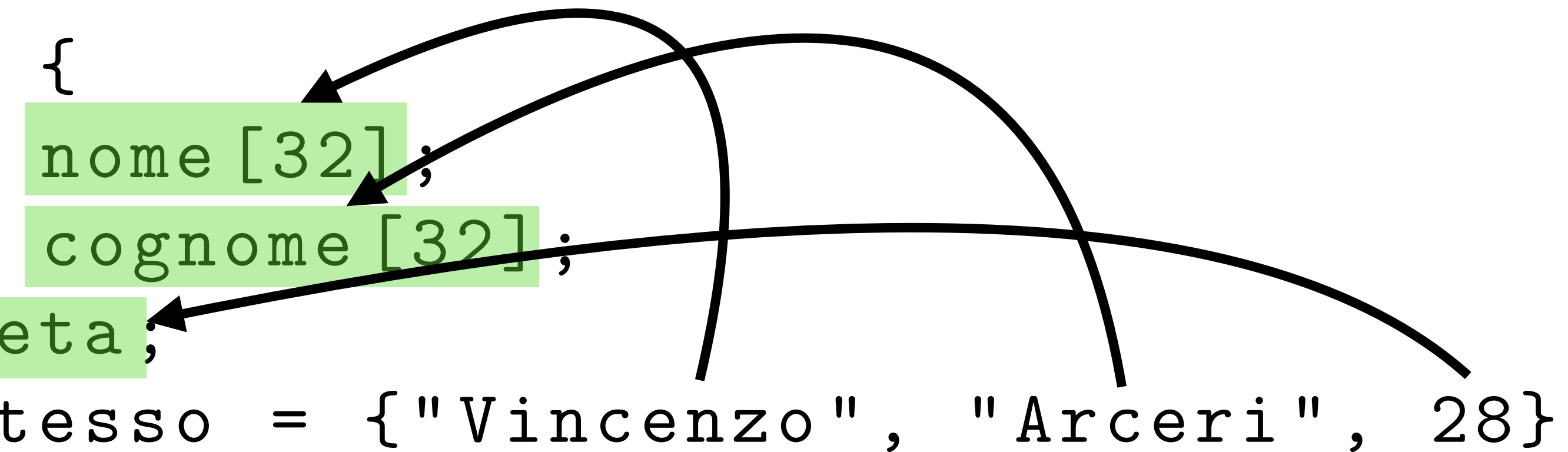
nome	?
cognome	?
eta	?

- Dichiarazione della variabile `me_stesso` del tipo struct costituito da tre campi
 - campo `nome` di tipo `char[32]`
 - campo `cognome` di tipo `char[32]`
 - campo `eta` di tipo `int`

Struct

Dichiarazione di una variabile di tipo struct con inizializzazione

```
struct {  
    char nome[32];  
    char cognome[32];  
    int eta;  
} me_stesso = {"Vincenzo", "Arceri", 28}
```



me_stesso

nome	"Vincenzo"
cognome	"Arceri"
eta	28

Struct

Definizione di un nuovo tipo

- In un unico comando, stiamo introducendo contemporaneamente un nuovo tipo e una nuova variabile di quel tipo

```
struct {  
    char nome[32];  
    char cognome[32];  
    int eta;  
} me_stesso;
```

Struct

Definizione di un nuovo tipo

- In un unico comando, stiamo introducendo contemporaneamente un nuovo tipo e una nuova variabile di quel tipo

```
struct {  
    char nome[32];  
    char cognome[32];  
    int eta;  
} me_stesso;
```


- Meglio dividere queste due operazioni

Struct

Definizione di un nuovo tipo

```
struct persona {  
    char nome[32];  
    char cognome[32];  
    int eta;  
};
```

Definizione di un nuovo tipo struct
composto da ... chiamato
persona



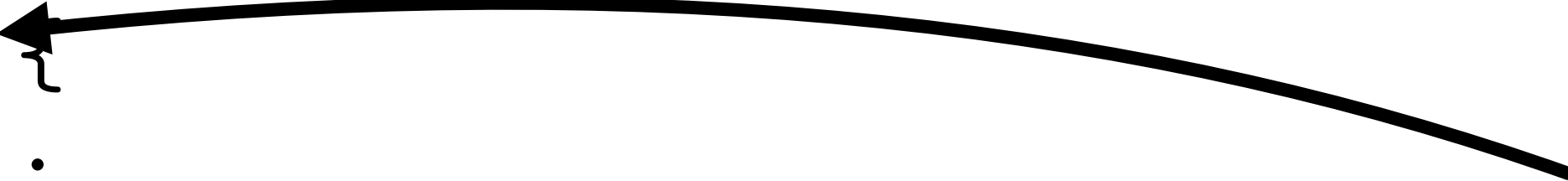
```
int main() {  
    persona me_stesso = {"Vincenzo", "Arceri", 28};  
    return 0;  
}
```

Struct

Definizione di un nuovo tipo

```
struct persona {  
    char nome[32];  
    char cognome[32];  
    int eta;  
};
```

L'identificatore `persona` è uno
shortcut usato per riferirsi al tipo
struct



```
int main() {  
    persona me_stesso = {"Vincenzo", "Arceri", 28};  
    return 0;  
}
```

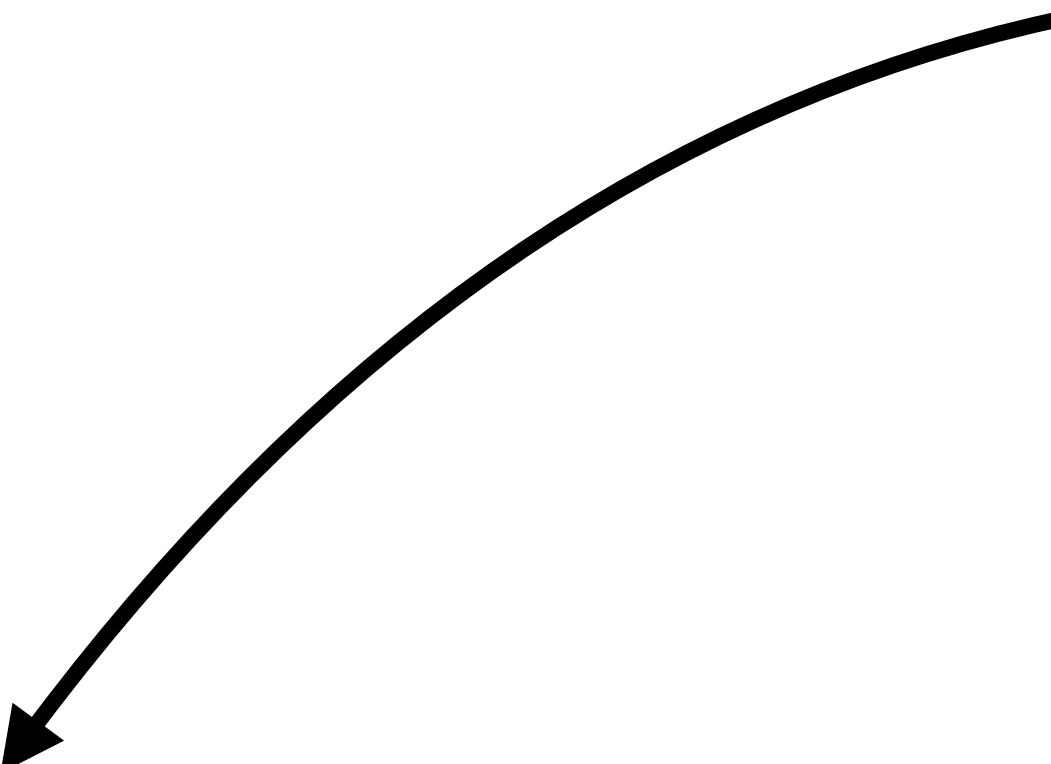
Struct

Definizione di un nuovo tipo

```
struct persona {  
    char nome[32];  
    char cognome[32];  
    int eta;  
};
```

```
int main() {  
    persona me_stesso = {"Vincenzo", "Arceri", 28};  
    return 0;  
}
```

Dichiarazione della variabile
me_stesso di tipo persona
(con inizializzazione)



Struct

Definizione di un nuovo tipo

```
#include <iostream>
using namespace std;

struct persona {
    char nome[32];
    char cognome[32];
    int eta;
};

int main() {
    persona me_stesso = { "Vincenzo", "Arceri", 28 };
    persona altra_persona = { "Mario", "Rossi", 52 };
    persona sconosciuto;
    return 0;
}
```

me_stesso

nome	“Vincenzo”
cognome	“Arceri”
eta	28

altra_persona

nome	“Mario”
cognome	“Rossi”
eta	52

sconosciuto

nome	?
cognome	?
eta	?

Struct

Selezione di un campo

- Supponiamo di avere il seguente tipo struct

```
struct nome {  
     $t_0$   $id_0$  ;  
     $t_1$   $id_1$  ;  
     $t_2$   $id_2$  ;  
    ...  
     $t_n$   $id_n$  ;  
};
```

- Una variabile x di tipo $nome$
- L'espressione $x . id_0$ accede al campo id_0 della struttura x