

# PROGRAMMAZIONE 2: SPERIMENTAZIONI

---

## Lezione 1 – Puntatori in C



# Agenda

- Introduzione
- Definire e inizializzare un puntatore
  - Dichiarare un puntatore
  - Inizializzare e assegnare valori al puntatore
- Operatori per i puntatori
  - Operatore di indirizzo &
  - Rappresentazione di un puntatore in memoria
  - Operatore di indirezione \*
  - Gli operatori & e \*
- Esempi con puntatori



Video lezione disponibile su YouTube: <https://youtu.be/jhpNgNzr10A>

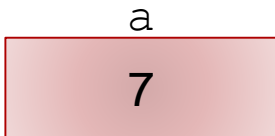
# Introduzione

- I **puntatori** sono uno dei costrutti più potenti del linguaggio di programmazione C.
- I puntatori fanno parte delle funzionalità del C più difficili da padroneggiare.
- Essi permettono di:
  - realizzare il passaggio per riferimento;
  - passare funzioni a funzioni;
  - creare e manipolare strutture dinamiche di dati, ossia quelle che possono crescere e contrarsi al momento dell'esecuzione del programma come **liste collegate**, code, pile e alberi.

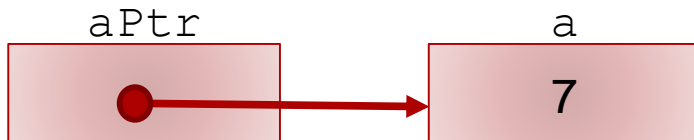
# Definire e inizializzare un puntatore

- I puntatori sono variabili i cui valori sono indirizzi di memoria.
- Normalmente una variabile contiene un valore specifico (es.: `int a = 7; float b = 4.5;`).
- Un puntatore, invece, contiene l'**indirizzo di memoria di una variabile** (variabile *puntata*).
- Un puntatore fa riferimento **indirettamente** al valore contenuto nella variabile puntata.
- Far riferimento a un valore per mezzo di un puntatore si dice **indirezione**.

# Definire e inizializzare un puntatore



Il nome `a` si riferisce direttamente a una variabile che contiene il valore 7.



Il puntatore `aPtr` si riferisce **indirettamente** a una variabile che contiene il valore 7 (**indirezione**).

# Dichiarare un puntatore

- I puntatori come tutte le altre variabili **devono essere dichiarati prima di essere utilizzati.**
- La definizione

```
int *aPtr, a;
```

- specifica che la variabile `aPtr` è di tipo `int *` (cioè un puntatore a un intero);
- il simbolo `*` si applica solamente alla variabile `a` che lo segue.

# Dichiarare un puntatore

- Come visto in precedenza, l'asterisco non viene distribuito a tutte le variabili dichiarate.
- Ogni variabile di tipo puntatore deve essere dichiarata con il simbolo `*` prefissato al nome.
- Un puntatore può *puntare* a variabili dello stesso tipo per cui è stato dichiarato (es.: **un puntatore a interi può puntare solo variabili intere**).
- Es.:
  - `int *p1, p2;`  
*// dichiara una sola variabile di tipo puntatore (p1)*
  - `int *p1, *p2;`  
*// dichiara due variabili di tipo puntatore (p1 e p2)*

# Dichiarare un puntatore



Si consiglia di includere le lettere `Ptr` nei nomi delle variabili di tipo puntatore.



Per evitare ambiguità, è bene dichiarare in modo separato i puntatori dai non puntatori.

Esempio:

```
int *aPtr, *bPtr;    // puntatori a variabili intere
int a, b;            // variabili intere
```



# Inizializzare un puntatore


- I puntatori **devono essere inizializzati** quando vengono definiti.
- Un puntatore può essere inizializzato a **NULL** o a un **indirizzo di memoria**.
- Un puntatore con il valore `NULL` indica che non sta puntando *niente* (ovvero *non sta puntando nessuna locazione di memoria*).
  - `NULL` è una *costante simbolica* definita in varie intestazioni (librerie) tra cui `<stdio.h>` che indica il valore 0.
  - Il valore 0 (o `NULL`) è l'unico valore intero che si può assegnare direttamente a una variabile di tipo puntatore.

# Operatori per i puntatori

- L'**operatore di indirizzo** `&` è un operatore unario che **restituisce l'indirizzo di memoria** del suo operando (il nome **della variabile**).
- Data, ad esempio, una variabile `a`, `&a` **restituisce l'indirizzo in memoria della variabile** `a`.

# Operatori per i puntatori

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int a = 7;
6      printf("\n\nTest sui puntatori in C \n\n");
7      printf("L'indirizzo di memoria di a e': %p \n\n", &a);
8      // &a mostra l'indirizzo di memoria di a
9      // %p mostra un risultato in forma esadecimale
10
11     return 1;
12 }
```

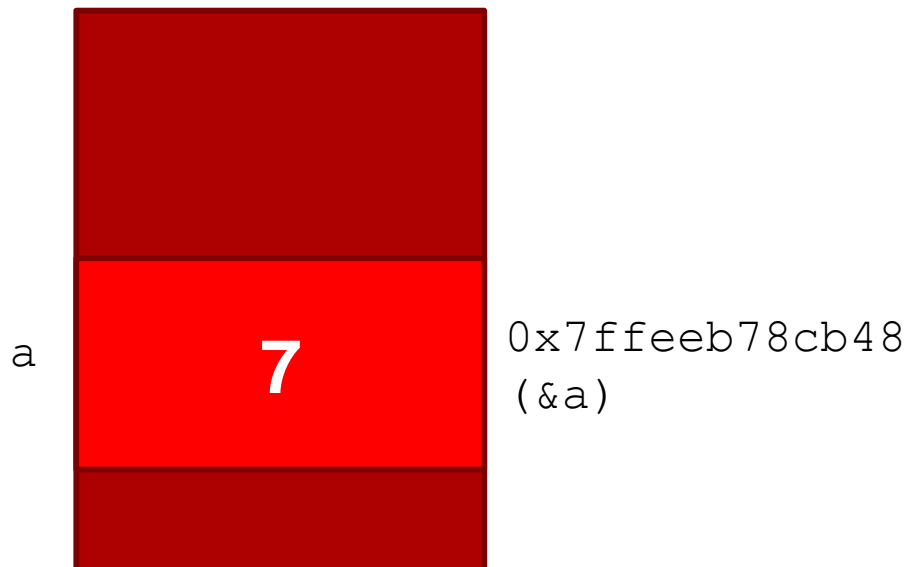


# Operatori per i puntatori

Test sui puntatori in C

L'indirizzo di a e': 0x7ffeeb78cb48

Memoria centrale (RAM)



# Operatori per i puntatori

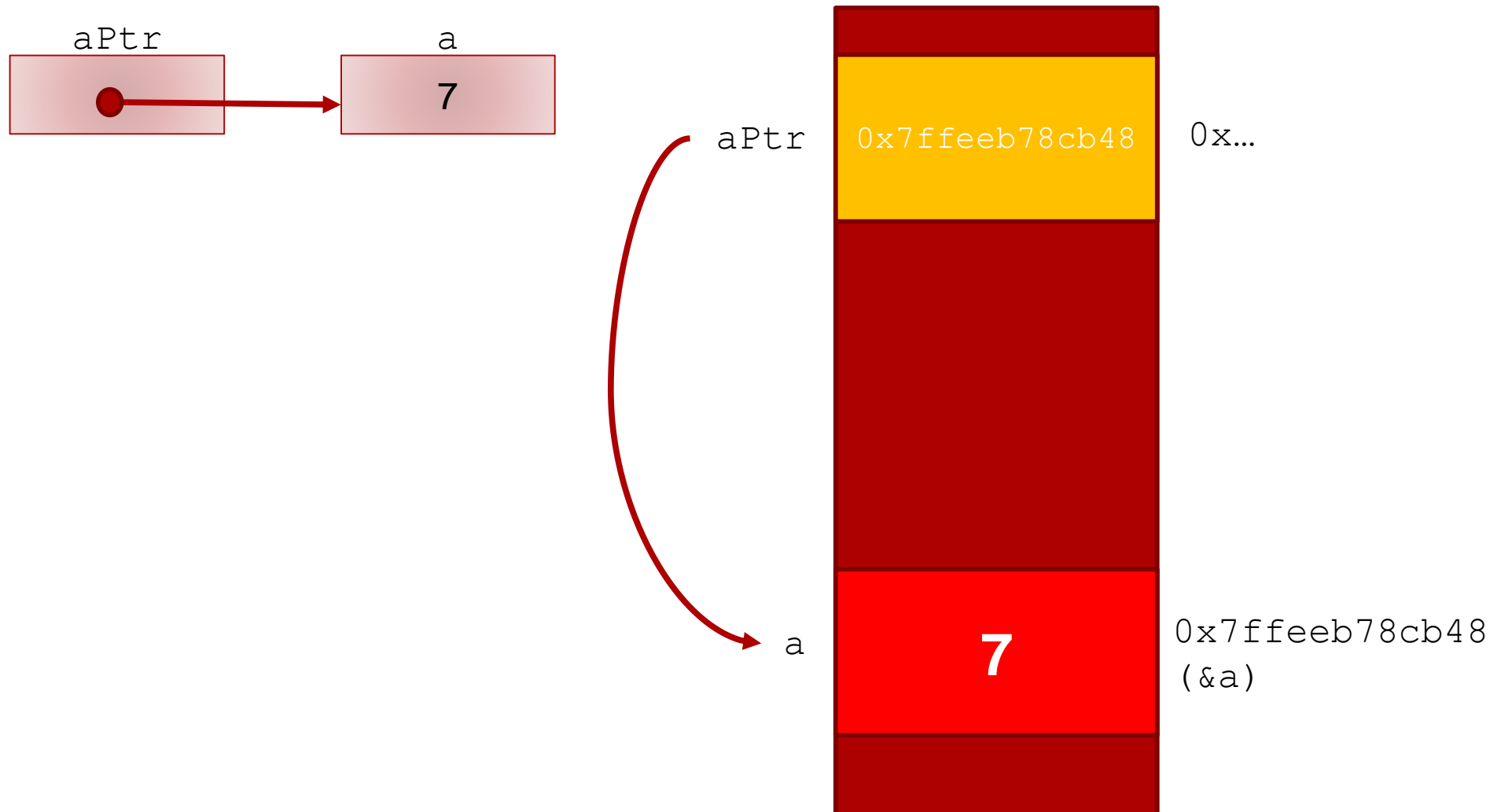
- Utilizzando l'operatore `&` è possibile, quindi, associare a un puntatore l'indirizzo di memoria di una variabile che si vuol far puntare.
- L'assegnazione nella riga 15 del codice indica dunque che il puntatore `aPtr` **punta ad** `a`.

```

10
11 // dichiara un puntatore a una variabile intera
12 int *aPtr;
13
14 // il puntatore punta all'indirizzo di memoria di a
15 aPtr = &a;
```



# Operatori per i puntatori




# Operatori per i puntatori

- **L'operatore di indirezione (o operatore di deferenziazione)  $*$  restituisce il valore contenuto dalla variabile alla quale sta *puntando* la variabile di tipo puntatore.**

# Operatori per i puntatori

- Se si effettua, ad esempio, la `printf` di `*aPtr`, si visualizza il contenuto della variabile puntata da `aPtr`.

```
11 // dichiara un puntatore a una variabile intera
12 int *aPtr;
13
14 // il puntatore punta all'indirizzo di memoria di a
15 aPtr = &a;
16
17 // indirezione
18 printf("Il contenuto puntato da aPtr e': %d \n\n", *aPtr);
```

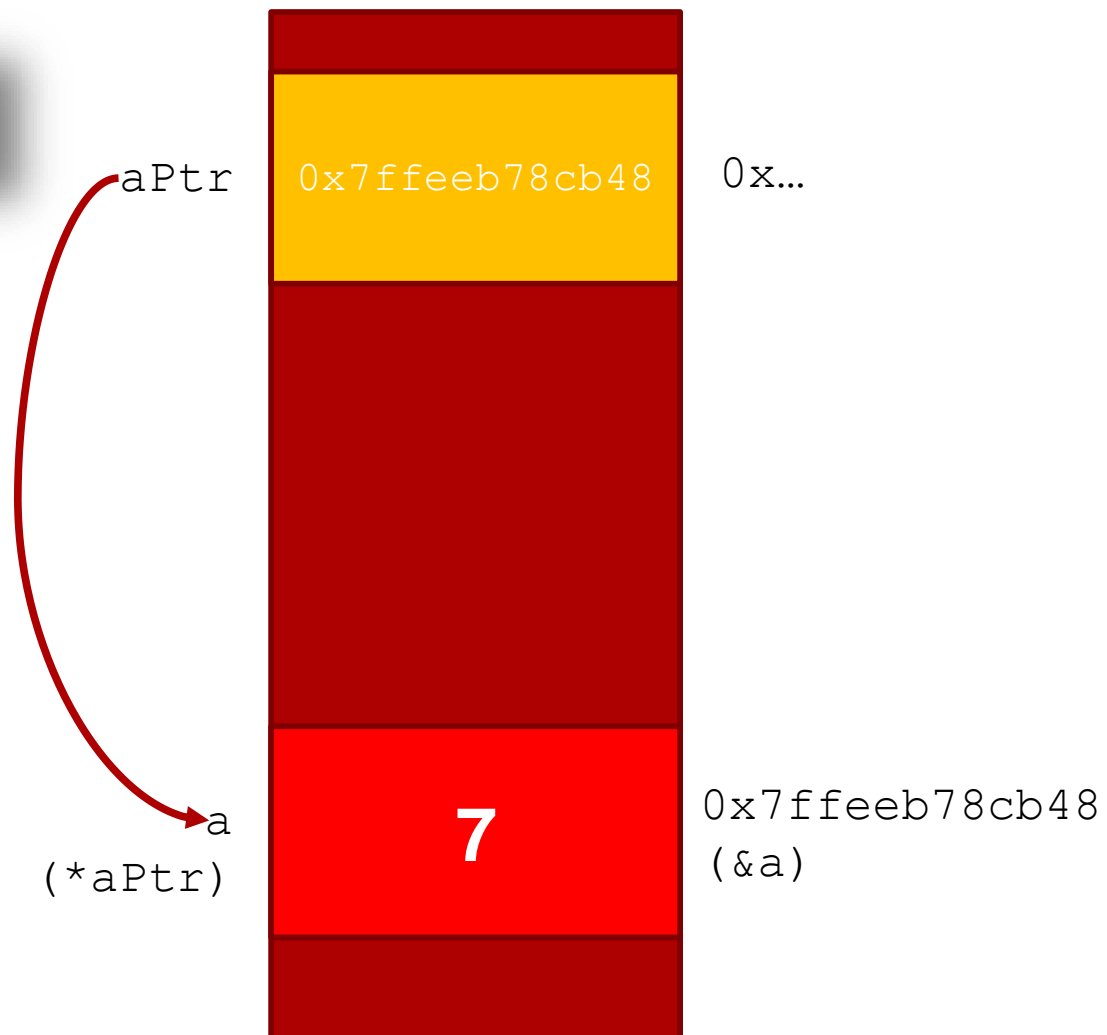




# Operatori per i puntatori

L'indirizzo di memoria di a e': 0x7ffee9fc2b48

Il contenuto puntato da aPtr e': 7



# Esempi con puntatori

- Esempio 1

```
int x=1, y=2;
```

```
int *p;
```

```
p = &x;
```

```
// p punta a x
```

```
y = *p;
```

```
// ad y assegna lo stesso valore puntato da p → quindi y = 1
```

# Esempi con puntatori

- Esempio 2

```
int x=1, y=2;
```

```
int *p;
```

```
p = &x;
```

```
// p punta a x
```

```
*p = *p + 5;
```

```
// all'indirizzo puntato da p aggiunge 5 → quindi x = 6
```

# Esempi con puntatori

- Esempio 3

```
int x=1, y=2;
```

```
int *p;
```

```
p = &x;
```

```
// p punta a x
```

```
p = p + 5;
```

```
// ERRORE! Cerca di modificare il valore dell'indirizzo di  
memoria contenuto in p (non possibile)
```

# Esempi con puntatori

- Esempio 4

```
int x=1;  
float y=5.5;  
int *p;
```

```
p = &y;
```

```
// ERRORE! p può puntare solo ad interi mentre y è un  
decimale
```

# Esempi con puntatori

- Esempio 5

```
int x=1, y=2;  
int *p;
```

```
p = &x;  
// p punta a x
```

```
*p++;  
// ERRORE! Non aumenta il valore puntato da p di 1 (a parità di priorità  
restituisce prima l'indirizzo di memoria poi lo aumenta quindi cerca di  
modificare l'indirizzo di memoria contenuto in p)
```

```
(*p)++;  
// Aumenta il valore puntato da p di 1 → quindi x = 2  
// equivale a *p = *p + 1;
```

# Esempi con puntatori

- Esempio 6: **vettori**.
- Il livello più alto di accesso ai dati tramite puntatori è **concesso da un puntatore non costante a dati non costanti**.
  - I dati possono essere modificati per mezzo del puntatore deferenziato e il puntatore può essere modificato per puntare ad altri dati.
- Un esempio di tale puntatore può essere utilizzato per gestire vettori e stringhe.

# Esempi con puntatori

- Esempio 6: vettori. Quando si lega un puntatore a un vettore lo si fa associando al puntatore la prima cella del vettore (la `[0]`).

```
int v[10];
int *p;

p = v;
// p punta al primo elemento del vettore v quindi a v[0]
// equivale a p = &v[0];

*p = 5;
// all'indirizzo puntato da p assegna 5 quindi v[0] = 5

printf("v[0] vale: %d \n", *p); // equivale a v[0]

p = p + 1; // sposta il puntatore all'elemento successivo (quindi a [1])
// equivale a ++p;

*p = 10; // imposta il contenuto puntato da p a 10

printf("v[1] vale: %d \n", v[1]);
// v[1] vale 1
```



# Esempi con puntatori

- Esempio 7: **stringhe**.
- Le stringhe sono vettori di caratteri terminati dal carattere speciale `'\0'`.
- Si può quindi ricondurre la gestione delle stringhe con i puntatori alla gestione dei vettori con i puntatori.

'C'	'i'	'a'	'o'	'\0'
[ 0 ]	[ 1 ]	[ 2 ]	[ 3 ]	[ 4 ]

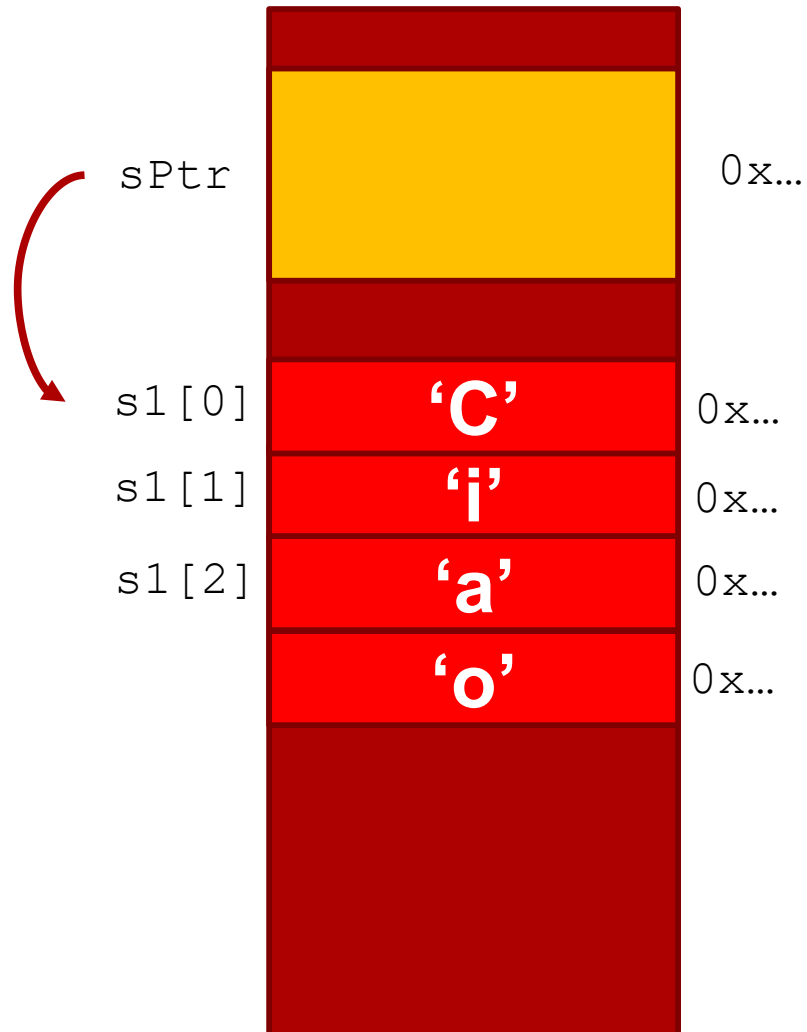
# Esempi con puntatori

- Come si può notare nel codice seguente viene dichiarato un puntatore a `char` che punterà la stringa `s1` (un vettore di caratteri).
- Il puntatore quindi punterà alla prima posizione (la `[0]`) della stringa.

```
5  int main(void)
6  {
7      // dichiara una stringa
8      char s1[] = "Ciao amici miei";
9      // s1[0] --> 'C', s1[1] --> 'i', s1[2] --> 'a', ecc...
10
11     // dichiara un puntatore a caratteri
12     char *sPtr;
13
14     sPtr = s1; // assegna al puntatore sPtr l'elemento di posizione [0] di s1
15
16     // mostra la stringa in maniera diretta
17     printf("La stringa contiene: %s\n\n", s1);
18
19     // mostra il contenuto puntato da sPtr (deferenzione)
20     printf("Elemento 0 della stringa: %c\n\n", *sPtr);
```


# Esempi con puntatori

```
sPtr = s1;
```



# Esempi con puntatori

- Come si può notare nella riga 24, è possibile modificare il contenuto di un elemento della stringa tramite puntatore.
- `(*sPtr)` accede al contenuto della stringa (un char), `++` aumenta tale contenuto di 1;
  - L'operazione equivale a `*sPtr = *sPtr + 1;`



```

16 // mostra la stringa in maniera diretta
17 printf("La stringa contiene: %s\n\n", s1);
18
19 // mostra il contenuto puntato da sPtr (deferenzione)
20 printf("Elemento 0 della stringa: %c\n\n", *sPtr);
21
22 // prende per deferenzione il contenuto puntato e lo aumenta di 1
23 // (quindi da C a D secondo la tabella ASCII)
24 (*sPtr)++;
25 printf("Elemento 0 della stringa modificato: %c\n\n", *sPtr);
26
27 ++sPtr; // sposta l'elemento puntato di una posizione
28
29 // mostra il contenuto puntato da sPtr
30 printf("Elemento 1 della stringa: %c\n\n", *sPtr);
31
32 ++sPtr; // sposta l'elemento puntato di una posizione
33
34 // mostra il contenuto puntato da sPtr
35 printf("Elemento 2 della stringa: %c\n\n", *sPtr);
36
37 // mostra la stringa (modificata tramite sPtr nella posizione 0)
38 printf("La stringa contiene: %s\n\n", s1);
    
```

# Esempi con puntatori

- Come si può notare nelle righe 27 e 32, è possibile spostare il puntatore per farlo puntare alle posizioni successive della stringa.
- L'operazione equivale a  $sPtr = sPtr + 1$ .

```
16 // mostra la stringa in maniera diretta
17 printf("La stringa contiene: %s\n\n", s1);
18
19 // mostra il contenuto puntato da sPtr (deferenzione)
20 printf("Elemento 0 della stringa: %c\n\n", *sPtr);
21
22 // prende per deferenzione il contenuto puntato e lo aumenta di 1
23 // (quindi da C a D secondo la tabella ASCII)
24 (*sPtr)++;
25 printf("Elemento 0 della stringa modificato: %c\n\n", *sPtr);
26
27 ++sPtr; // sposta l'elemento puntato di una posizione
28
29 // mostra il contenuto puntato da sPtr
30 printf("Elemento 1 della stringa: %c\n\n", *sPtr);
31
32 ++sPtr; // sposta l'elemento puntato di una posizione
33
34 // mostra il contenuto puntato da sPtr
35 printf("Elemento 2 della stringa: %c\n\n", *sPtr);
36
37 // mostra la stringa (modificata tramite sPtr nella posizione 0)
38 printf("La stringa contiene: %s\n\n", s1);
```

# Esempi con puntatori

- Esempio 8: **stringhe e funzioni.**
- Ricevere una stringa (o un vettore) come argomento di una funzione che elabora e modifica ogni carattere nella stringa trasformandolo in maiuscolo.

# Esempi con puntatori

- La stringa, essendo un vettore, viene sempre passata alla funzione per riferimento (quindi viene passato il puntatore al suo primo elemento, quello di posizione `[0]`).

```
8  int main(void)
9  {
10     char s1[] = "Ciao amici miei";
11     printf("\nConversione di una stringa\n\n");
12     printf("Stringa iniziale: %s\n\n", s1);
13     converti(s1);
14     printf("Stringa convertita: %s\n\n", s1);
15     return 1;
16 }
```

# Esempi con puntatori

- La funzione ha un parametro formale che è un puntatore a un carattere; il puntatore punterà al primo carattere della stringa (posizione `[0]`).
- `*sPtr` conterrà ogni volta il carattere corrente della stringa quindi ogni modifica applicata tramite `*sPtr` comporta la modifica del carattere nella stringa.

```

18 void converti(char *sPtr)
19 {
20     while(*sPtr!='\0') // il contenuto puntato è diverso da '\0'
21     {
22         *sPtr = toupper(*sPtr); // toupper definita in ctype.h
23         ++sPtr; // aumenta il puntatore alle posizioni della stringa di 1 (si passa da [0] a [1] ecc...)
24     }
25 }

```

`toupper()` è una funzione che ha in input un carattere e lo trasforma in maiuscolo



# FINE PRESENTAZIONE

---

