Laboratorio di Programmazione e Strutture dati a.a. 2022/2023

Fabio Narducci

Alcuni cenni a stringhe, puntatori e allocazione dinamica

Le stringhe in C

- Una stringa è una sequenza di caratteri che termina con un carattere speciale di fine stringa **\0**
- E' improprio pensare alla stringa come un semplice array di caratteri in quanto c'è una differenza lieve tra i due:
  - Array
  - Stringa

C	**a**	**s**	**a**		
C	**a**	**s**	**a**	**\0**	

- In riferimento al numero di caratteri allocati da un array di caratteri e una stringa c'è una differenza anche se apparentemente sono identiche

```
#include <stdio.h>
#include <string.h>

int main (){
char array[4]={'C', 'a', 's', 'a'};

char stringa[] ="Casa";
printf("%d, %d, %d\n",
           (int)sizeof(array),
           (int)sizeof(stringa),
           (int)strlen(stringa));
}
```

Dichiarazione e lettura/scrittura di una stringa

- Dichiarare/Inizializzazione:
  - `char array[20]; //dichiaro un array di 20 caratteri non inizializzato`
  - `char stringa[]="Ciao"; //dichiaro ed inizializzo lo spazio strettamente necessario`

- Scrivere in una stringa
  - `scanf("%s", stringa);`

- Leggere una stringa
  - `printf("%s", stringa);`

Di più sulla lettura/scrittura delle stringhe

- La lettura da scanf può talvolta metterci in difficoltà con la lettura di stringhe composte da spazi.
- Per risolvere questa necessità si può ricorrere alle seguenti funzioni standard:
  - **GETS**: `char * gets ( char * str ); //lancia un warning`
  - **FGETS**: `char * fgets ( char * str, int num, FILE * stream );`
- Le due funzioni *gets* e *fgets* leggono stringhe da un file (implicitamente nella gets) che può essere impostato per *fgets* su *stdin (standard input).*
- Le due funzioni restituiscono un numero non-negativo in caso di successo e EOF in caso di errore.

Funzioni utili per le stringhe (*string.h*)

```
char stringa1[]="Ciao";
char stringa2[]="Mondo";
```

- **Lunghezza**
  - `strlen(stringa1)` -> 4 (escluso lo *0*)
  - sizeof(`stringa1/sizeof(stringa1[0])`) - > 5 (incluso lo \0)

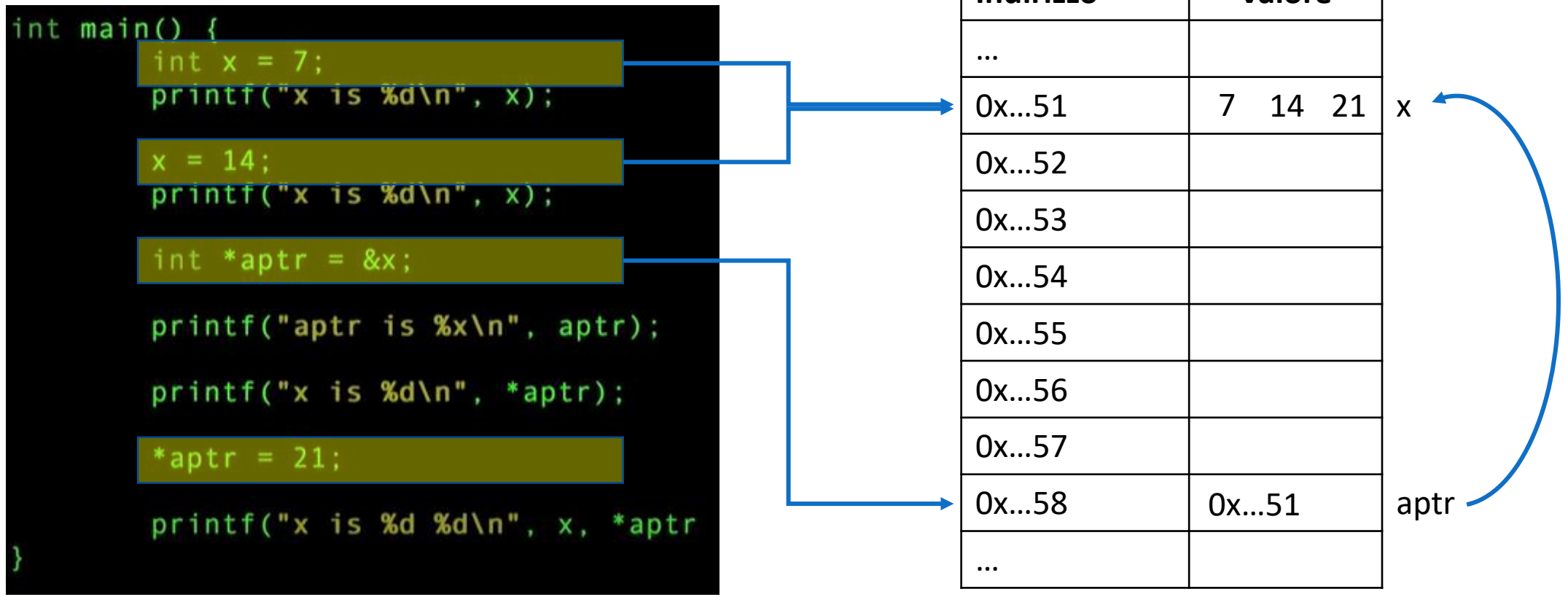
- **Confronto**
  - `strcmp(stringa1, stringa2)` -> $\begin{cases} 0 & se\ uguali \\ < 0 & se\ il\ primo\ carattere\ diverso\ di\ stringa1 <_{ASCII}\ del\ corrispondente\ di\ stringa2 \\ > 0 & se\ il\ primo\ carattere\ diverso\ di\ stringa1 >_{ASCII}\ del\ corrispondente\ di\ stringa2 \end{cases}$

- **Copia**
  - `strcpy(destinazione, sorgente)` -> copia il contenuto di sorgente in destinazione

- **Concatenazione**
  - `strcat(stringa1, stringa2)` -> concatena stringa2 a stringa1, quindi stringa1="CiaoMondo"

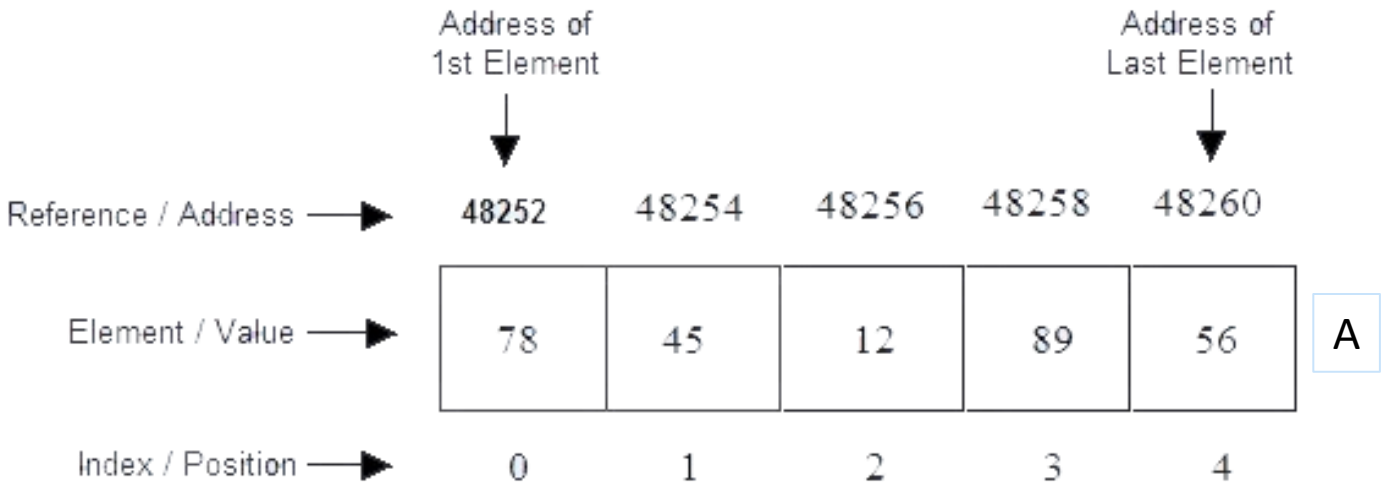
I puntatori

```
int main() {
        int x = 7;
        printf("x is %d\n", x);

        x = 14;
        printf("x is %d\n", x);

        int *aptr = &x;

        printf("aptr is %x\n", aptr);

        printf("x is %d\n", *aptr);

        *aptr = 21;

        printf("x is %d %d\n", x, *aptr
}
```



Puntatori e Array

- Il nome di un array è il puntatore all'area di memoria allocata per l'array
- Ecco perchè nel passaggio di parametri a funzione che siano array è sufficiente passare il nome dell'array!



A[2] = ?
A = ?
*A = ?
&A[1] = ?

malloc(), calloc(), realloc(), free()

- ptr = (cast-type*) **malloc**(byte-size)
  - *ptr_m = (int*) malloc(10 * sizeof(int));*
- ptr = (cast-type*)**calloc**(n, element-size);
  - *ptr_c = (float*) calloc(25, sizeof(float));*
- Nessuno ci garantisce che l'allocazione è andata a buon fine quindi è sempre opportuno controllare dopo una allocazione dinamica

```
if (ptr == NULL) {
        printf("Memoria non allocata.\n");
        exit(0);
    }
```

- ptr = realloc(ptr, newSize);
  - *ptr_m=realloc(prt_m, 100);*
- free(ptr);

ACCESSO ALLO SPAZIO CLOUD DEDICATO ALLE ATTIVITA' DI LABORATORIO

- **Turno 9-11:** shorturl.at/byMOV

- **Turno 11-13**: shorturl.at/beit5

FINE

Stringhe e puntatori