
Le stringhe

Definizione

- Una **stringa** è un array di caratteri terminato dal carattere nullo, corrispondente alla sequenza di escape `\0` (con valore numerico associato zero)
- Una **stringa costante** (o **letterale**) è una serie di caratteri racchiusi fra doppi apici: tale stringa è di tipo **array di caratteri**, con ogni carattere che occupa un byte
- Ad ogni stringa viene aggiunto automaticamente dal compilatore un carattere nullo, ad indicarne la fine

Dichiarazione e inizializzazione – 1

- ✦ Per memorizzare una stringa occorre dichiarare un array di **char**, che può essere inizializzato con una stringa costante
- ✦ L'array ha dimensione maggiore di uno rispetto alla lunghezza della stringa, per consentire la memorizzazione del carattere nullo di terminazione (**str** ha lunghezza 6 byte)
- ✦ Il compilatore segnala un errore se si dichiara la lunghezza della stringa n , e si inizializza con una stringa costante di lunghezza $>n$

```
static char str[3]="quattro"; /* SCORRETTO */
```

```
static char str1[3]="tre";    /* CORRETTO */
```

I compilatori ANSI, generalmente, consentono di specificare una dimensione di array che non includa il carattere terminatore

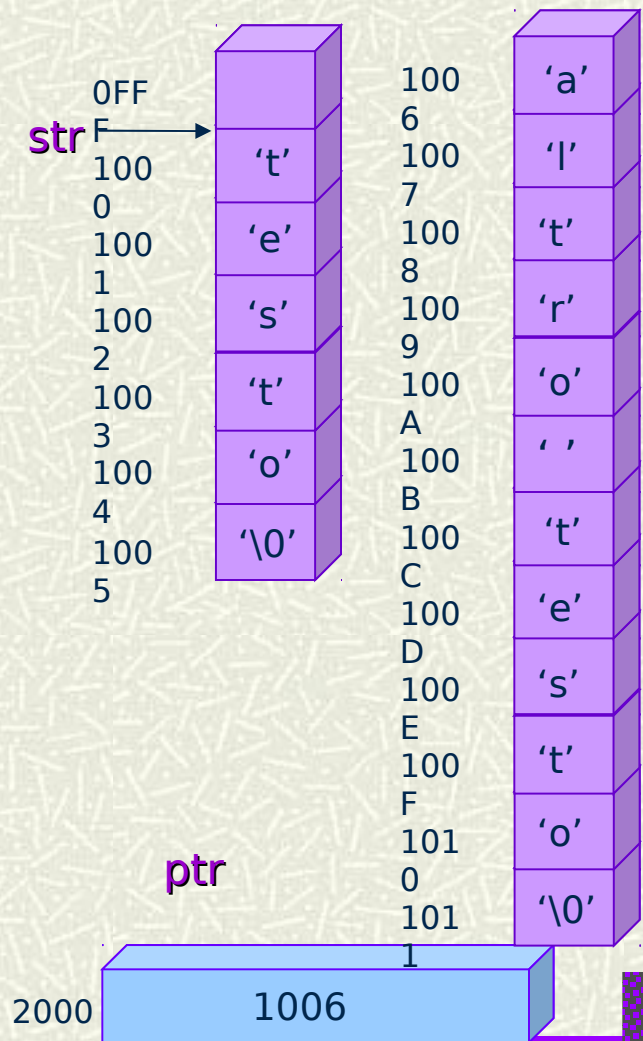
Dichiarazione e inizializzazione – 2

- È possibile inizializzare un puntatore a **char** con una stringa costante:

```
char *ptr = "altro testo";
```

si crea un array di caratteri, inizializzato ad "altro testo", riservando però memoria anche per il puntatore

- Nel caso dell'array, tutti i successivi accessi utilizzano il nome dell'array come riferimento per l'indirizzo dell'elemento iniziale dell'array: tale indirizzo non può essere modificato
- Il puntatore è una variabile e può essere modificato: l'indirizzo relativo alla prima inizializzazione viene perso



Gli assegnamenti a stringhe

- Un puntatore a **char** può essere inizializzato con una stringa costante, perché una stringa è un array di **char**
- Una stringa costante viene interpretata come un puntatore al primo carattere della stringa

```
#include <stdlib.h>
main()
{
    char array[10];
    char *ptr1 = "10 spazi";
    char *ptr2;

    array = "not OK"; /* non è possibile assegnare un indirizzo */
    array[5] = 'A';    /* OK */
    *(ptr1+5) = 'B';   /* OK */
    ptr1 = "OK";
    ptr1[5] = 'C';     /* opinabile a causa dell'assegnamento
precedente */
    *ptr2 = "not OK"; /* conflitto di tipi */
    ptr2 = "OK";
    exit(0);
}
```

Stringhe e caratteri – 1

- ✦ Occorre notare la differenza fra stringhe costanti e costanti di tipo carattere:

```
char ch = 'a'; /* Per 'a' è riservato un byte */
```

```
/* Vengono riservati due byte per "a",  
oltre allo  
* spazio necessario alla memorizzazione  
di ps  
*/  
char *ps = "a";
```

- ✦ È possibile assegnare una costante carattere all'indirizzo contenuto in un puntatore a **char**; è invece scorretto effettuare la stessa operazione relativamente ad una stringa

```
char *p1;  
*p1 = 'a'; /* OK */  
*p1 = "a"; /* not OK  
*/
```

```
char *p2;  
p2 = 'a'; /* not OK  
*/  
p2 = "a"; /* OK */
```

Le stringhe sono interpretate come puntatori a carattere

Stringhe e caratteri – 2

- Le inizializzazioni e gli assegnamenti non sono simmetrici; è infatti possibile scrivere

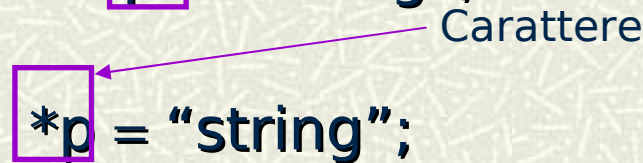
`char *p = "string";`



Puntatore a carattere

ma non...

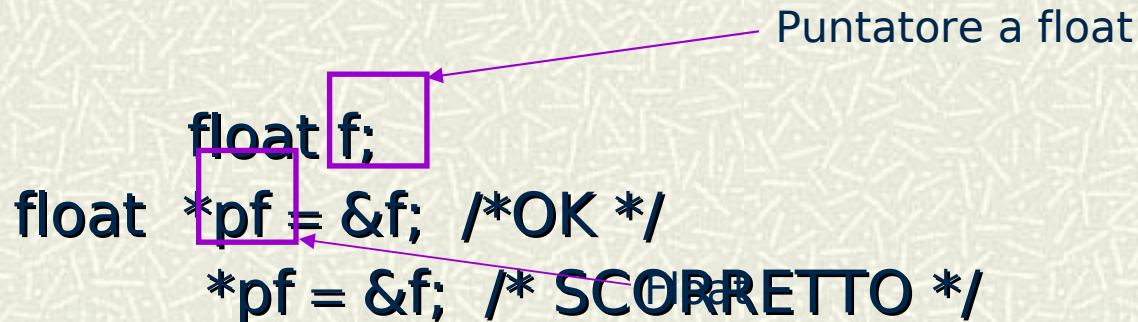
`*p = "string";`



Carattere

- Nota:** vale per inizializzazioni ed assegnamenti di tutti i tipi di dati

`float f;`



Puntatore a float

`float *pf = &f; /*OK */`

`*pf = &f; /* SCORRETTO */`

Lettura e scrittura di stringhe – 1

- Le stringhe possono essere lette e scritte utilizzando le funzioni ***scanf()*** e ***printf()***, con lo specificatore di formato ***%s***
- L'argomento della funzione ***scanf()*** deve essere un puntatore ad un array di caratteri di dimensioni sufficienti a contenere la stringa in ingresso, che si intende terminata da un qualsiasi carattere di spaziatura
- La funzione ***scanf()***, dopo aver letto il dato in ingresso, aggiunge automaticamente il carattere ***\0*** in fondo alla stringa
- L'argomento della funzione ***printf()*** deve essere un puntatore ad un array di caratteri terminato dal carattere nullo (che non viene stampato)

Lettura e scrittura di stringhe – 2

- **Esempio:** Scrivere un programma che legge una stringa dalla periferica d'ingresso di default e la

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_CHAR 80

main()
{
    char str[MAX_CHAR];
    int i;

    printf("Introdurre una stringa:");
    scanf("%s", str);
    for (i=0; i<10; i++)
        printf("%s\n", str);
    exit(0);
}
```

È possibile utilizzare il nome dell'array come argomento per le funzioni di I/O, in quanto puntatore all'inizio dell'array

Le funzioni di libreria per le stringhe:

strlen()

- La funzione *strlen()*, restituisce il numero di caratteri che compongono una stringa (escluso il carattere nullo)

- Poiché nell'espressione **str++* i due operatori hanno la stessa precedenza ed associatività destra, l'espressione viene analizzata dal compilatore nel modo seguente:

- Valutazione dell'operatore di incremento postfisso; il compilatore passa *str* all'operatore successivo e lo incrementa solo al termine della valutazione dell'espressione

- Valutazione dell'operatore ***, applicato a *str*

- Completamento dell'espressione, con l'incremento di *str*

```
int strlen(str)
char *str;
{
    int i;
    for (i=0; *str++; i++)
        ; /* istruzione vuota */
    return i;
}
```

Le funzioni di libreria per le stringhe:

strcpy()

- # La funzione *strcpy()* copia una stringa in un'altra
- # Il risultato dell'assegnamento costituisce la condizione di test per il ciclo **while**
- # Se **s2* vale zero (per il carattere di terminazione), si ha l'uscita dal ciclo
- # L'operatore di incremento postfisso è obbligatorio: un incremento prefisso non produrrebbe un risultato corretto, dato che il primo elemento non verrebbe copiato

```
void strcpy(s1, s2)
char *s1, *s2;
{
    while (*s2++ = *s1++)
        ; /*istruzione vuota*/
}
```


Le funzioni di libreria per le stringhe:

strstr() – 1

- La funzione *strstr()* effettua la ricerca di una sottostringa all'interno di una stringa, operazione detta comunemente *pattern matching*
- La funzione prevede come argomenti due puntatori a stringhe di caratteri ed effettua la ricerca di un'occorrenza della seconda stringa nella prima:
 - se esiste un'occorrenza, viene restituita la posizione d'inizio nell'array
 - altrimenti, viene restituito -1
- **Nota:** la maggior parte delle funzioni della libreria di run-time restituisce 0 o -1, come valore di errore (per *strstr()*, 0 corrisponde all'occorrenza della seconda stringa all'inizio della prima)

Le funzioni di libreria per le stringhe:

strstr() – 2

```
/* Restituisce la posizione di str2 in str1; restituisce -1 se non esiste occorrenza */
int strstr(str1, str2)
char *str1, *str2;
{
    char *p, *q, *substr;

    /* Itera su ogni carattere di str1 */
    for (substr=str1; *substr; substr++)
    {
        p = substr;
        q = str2;
        /* Controlla se l'occorrenza di str2 corrisponde alla posizione corrente */
        while (*q)
            if (*q++ != *p++)
                goto no_match; /* serve per uscire dal while, ma restare nel
for */
        /* Si giunge qui solo se si è trovata un'occorrenza di str2 */
        return substr-str1;
    }
    /* Si giunge qui se non è stata riscontrata un'occorrenza di str2 (nel ciclo
while) */
    no_match: ;
}
/* Si giunge qui se non vi sono occorrenze di str2 in str1 */
return -1;
}
```

Le funzioni di libreria in string.h

| | |
|------------|---|
| strcpy() | Copia una stringa in un array |
| strncpy() | Copia una parte di una stringa in un array |
| strcat() | Concatena due stringhe |
| strncat() | Concatena parte di una stringa ad un'altra |
| strcmp() | Confronta due stringhe |
| strncmp() | Confronta due stringhe per una lunghezza data |
| strchr() | Cerca la prima occorrenza di un carattere specificato in una stringa |
| strcoll() | Confronta due stringhe sulla base di una sequenza di confronto definita |
| strcspn() | Calcola la lunghezza di una stringa che non contiene i caratteri specificati |
| strerror() | Fa corrispondere ad un numero di errore un messaggio di errore testuale |
| strlen() | Calcola la lunghezza di una stringa |
| strpbrk() | Cerca la prima occorrenza di uno tra i caratteri specificati all'interno di una stringa |
| strrchr() | Cerca l'ultima occorrenza di un carattere in una stringa |
| strspn() | Calcola la lunghezza di una stringa che contenga caratteri specificati |
| strstr() | Cerca la prima occorrenza di una stringa in un'altra |
| strtok() | Divide una stringa in una sequenza di simboli |
| strxfrm() | Trasforma una stringa in modo che sia utilizzabile come argomento per strcmp() |

Esempio: Calcolo dell'età

```
/* Esempio di conversione da stringa ad intero */
#include <stdio.h>
#include <stdlib.h>

main()
{
    char anno_nascita[5], anno_corrente[5];
    int anni;

    printf("Inserire l'anno di nascita: ");
    scanf("%s", anno_nascita);
    printf("Inserire l'anno corrente: ");
    scanf("%s", anno_corrente);

    /* atoi() converte una stringa in un intero */
    anni = atoi(anno_corrente) -
atoi(anno_nascita);
    printf("Età: %d\n", anni);
    exit(0);
}
```

Esempio: Parole nella stringa

```
/* Conta il numero di parole in una stringa */
#include <ctype.h>
int word_count(s)
char *s;
{
    int count=0;
    while (*s != '\0')
    {
        while (isspace(*s))    /* salta la spaziatura */
            ++s;
        if (*s != '\0')        /* trovata una parola */
        {
            ++count;
            while (!isspace(*s) && *s != '\0')
                ++s;            /* salta la parola */
        }
    }
    return count;
}
```

Esempio: Parole palindrome

```
/* Letta in input una stringa, verifica se è palindroma */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

main()
{
    char parola[32], i=0, n;
    printf("Inserisci una parola (lunga al max 31 caratteri): ");
    scanf("%s", parola);
    n = strlen(parola);
    while ((i <= n/2) && (parola[i] == parola[n-1-i]))
        i++;
    if (i > n/2)
        printf("La parola %s è palindroma.\n", parola);
    else
        printf("La parola %s non è palindroma.\n", parola);
    exit(0);
}
```


Esempio: Da minuscole a maiuscole

```
/* Letta in input una stringa alfabetica, la riscrive utilizzando solo lettere maiuscole */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

main()
{
    char s[100], t[100];
    int i;
    printf("Inserisci una stringa: ");
    scanf("%s", s);
    for (i=0; i<strlen(s); i++)
    {
        if (s[i] >= 97 && s[i] <= 122)
            t[i] = s[i] - 32;
        else
            t[i] = s[i];
    }
    printf("Stringa maiuscola: %s\n", t);
    exit(0);
}
```