
```
char* titolo=  
{'l','e',' ','s','t','r','i','n','g','h','e','\0'};
```

1

Libreria sulle stringhe

- La libreria **string.h** contiene una ricca serie di funzioni per operare sulle stringhe
- Esempi:
 - copiare una stringa in un'altra (**strcpy**)
 - concatenare due stringhe (**strcat**)
 - confrontare due stringhe (**strcmp**)
 - cercare un carattere in una stringa (**strchr**)
 - cercare una stringa in un'altra (**strstr**)
 - ...

2

Lunghezza di una stringa

```
int strlen(char* str);
```

- La funzione restituisce la lunghezza della stringa `str`, senza contare il terminatore

- Es:

```
char pippo[]={ 'p', 'i', 'p', 'p', 'o', '\0' };  
char pluto[50]="pluto";  
sizeof(pippo); ⇨ 6  
strlen(pippo); ⇨ 5  
sizeof(pluto); ⇨ 50  
strlen(pluto); ⇨ 5
```

3

Possibile implementazione

- Idea

- Scandire la stringa contando i caratteri
- Terminare il conteggio quando si incontra il terminatore `'\0'`

```
int strlen(char* str)  
{  
    int len = 0;  
    while(str[len] != '\0')  
        len++;  
    return len;  
}
```

*Accumulatore
per il conteggio*

4

Comparazione fra stringhe

```
int strcmp(char* str1, char* str2);
```

- La funzione restituisce un valore
 - <0 se `str1` precede lessicograficamente `str2`
 - >0 se `str1` segue lessicograficamente `str2`
 - =0 se `str1` e `str2` sono identiche

- Es:

```
char s1[]="pluto";  
char s2[]="plutone";  
char s3[]="zio paperino";  
strcmp(s1,s1); ⇨ 0  
strcmp(s1,s2); ⇨ <0  
strcmp(s3,s2); ⇨ >0
```

N.B.:

Esiste anche la funzione di libreria `strncmp`, che prende in input anche il massimo numero di caratteri che si vogliono confrontare

5

Possibile implementazione 1/3

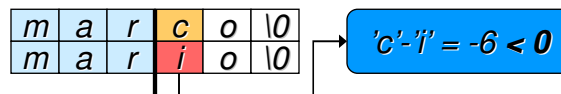
- Idea
 - Scandiamo parallelamente carattere per carattere le due stringhe confrontando di volta in volta i due caratteri trovati
 - Ci fermiamo quando vale una delle seguenti condizioni
 - I caratteri letti sono differenti
 - L'ordine lessicografico è determinato dall'ordine alfabetico dei due caratteri
 - Una delle due stringhe è giunta al termine (ovvero abbiamo incontrato il suo terminatore)
 - Ciò significa che le due stringhe sono identiche, oppure una è inclusa nell'altra

6

Possibile implementazione 2/3

■ ...idea (continua)

- Ricordiamoci che per valutare l'ordine alfabetico di due caratteri basta compararne i valori numerici (il terminatore vale 0)
- Quindi basta restituire la differenza tra la coppia di caratteri che si incontra alla fine della scansione
- Esempio:



7

Possibile implementazione 3/3

```
int strcmp(char* str1, char* str2)
{
    int i=0;
    while(str1[i] != '\0' &&
          str2[i] != '\0' &&
          str1[i]==str2[i])
        i++;
    return str1[i] - str2[i];
}
```

8

Concatenazione di stringhe

```
char* strcat(char* dest, char* src)
```

- La funzione appende la stringa `src` in coda alla stringa `dest`

- **Ipotesi:** la stringa `dest` ha abbastanza spazio per contenere anche `src`

- Viene restituito il puntatore a `dest`

- Es:

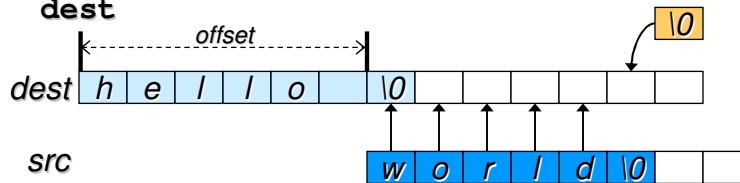
```
char s1[50] = "hello ";  
char* s2 = "world!";  
printf("%s", strcat(s1, s2)); ⇨ hello world!
```

9

Possibile implementazione 1/2

- Idea

- Ci posizioniamo sul terminatore della stringa `dest`
 - È l'elemento di posizione `strlen(dest)`
- A partire da questo punto, copiamo il contenuto della seconda stringa carattere per carattere, fino a quando non raggiungiamo il suo terminatore
- Infine, appendiamo un nuovo terminatore alla stringa `dest`



10

Altra implementazione 2/2

- Dobbiamo ricordarci che non possiamo semplicemente restituire il valore di `strcpy`
 - È il puntatore alla posizione in cui abbiamo copiato `src`

```
char* strcat(char* dest, char* src)
{
    strcpy(dest + strlen(dest), src);
    return dest;
}
```

13

Ricerca di un carattere in una stringa

```
char* strchr(char* str, int car);
```

- La funzione ricerca il carattere `car` all'interno della stringa `str` (compreso il terminatore) e restituisce
 - La posizione della prima occorrenza del carattere nella stringa
 - `NULL` se il carattere non viene trovato

- Es:

```
char* find = strchr("hello world!", 'w');
printf("%s", find); ⇒ world!
```

14

Possibile implementazione 1/2

■ Idea

- Scandisco la stringa con un puntatore, finchè non raggiungo il terminatore
- Se il carattere alla posizione corrente è uguale a `car`, restituisco il puntatore corrente
- Se invece termino il ciclo di scansione senza successo, restituisco **NULL**

15

Possibile implementazione 2/2

```
char* strchr(char* str, int car)
{
    do
    {
        if(*str == car)
            return str;
    }
    while (*str++);
    return NULL;
}
```

Dereferenziazione
del puntatore

Prima viene testato se il valore
attualmente puntato da `str` non
è nullo (ovvero non corrisponde
al terminatore), poi il puntatore
viene incrementato per
continuare la scansione

16

Ricerca di una stringa in un'altra stringa

```
char* strstr(char* str, char* sub);
```

- La funzione ricerca la stringa `sub` all'interno della stringa `str`, restituendo
 - L'intera stringa `str` se `sub` è la stringa vuota
 - La posizione dell'inizio della prima occorrenza della sottostringa `sub` in `str`
 - `NULL` se la sottostringa non è presente all'interno di `str`

- Es:

```
char* result=strstr("Say bye bye!","bye");  
printf("%s",result); ⇨ bye bye!
```

17

Possibile implementazione 1/2

- Idea

- Individuo la prima occorrenza dell'iniziale di `sub` all'interno di `str`
 - Posso usare `strchr`
- Controllo se la parte di `str` che rimane ha come prefisso `sub`
 - Posso usare `strncmp`, utilizzando come numero di caratteri da confrontare la lunghezza di `sub`
- In caso affermativo, restituisco questa parte di `str`
- Altrimenti, proseguo ricercando la successiva occorrenza dell'iniziale di `sub`

18

Possibile implementazione 2/2

```
char* strstr(char *str, char* sub)
{
    int subLength=strlen(sub);
    char* p=str;
    for( ; (p=strchr(p, sub[0])) !=NULL; p++)
    {
        if(strncmp(p, sub, subLength)==0)
            //sub prefisso
            return p;
    }
    return NULL;
}
```

Cerca il carattere `sub[0]` in `p`, riassegna il risultato a `p`, poi testa che il valore non sia nullo

Necessario perché altrimenti `strchr` continua a trovare sempre la stessa occorrenza di `sub[0]`, senza spostarsi sulla successiva

19

La soluzione dell'autarchico... ...e della CRT

```
char* strstr (const char *str1, const char *str2)
{
    char *cp = (char*) str1;
    char *s1, *s2;
    if (!*str2)
        return((char*) str1);
    while (*cp)
    {
        s1 = cp;
        s2 = (char*) str2;
        while (*s1 && *s2 && !(*s1 - *s2))
            s1++, s2++;
        if (!*s2)
            return(cp);
        cp++;
    }
    return (NULL);
}
```

20