ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fondamenti di Informatica T

Stringhe in C

Stringhe in C

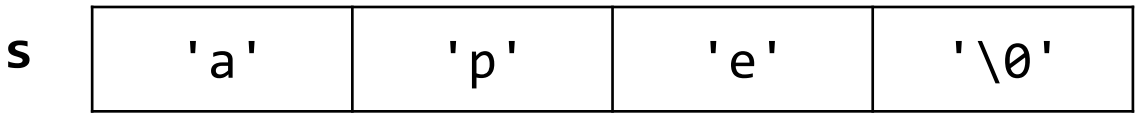
Per stringa si intende una sequenza di caratteri

E.g. "ciao mondo", "Gigi"...

In C, una stringa è un array di caratteri con una convenzione

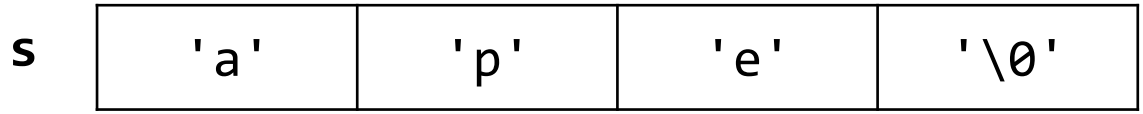
La convenzione: l'ultimo carattere è sempre '\0'

- Si chiama "terminatore"
- Esempio:



Stringhe in C

L'uso del terminatore:



...Permette di ricavare la dimensione della stringa!

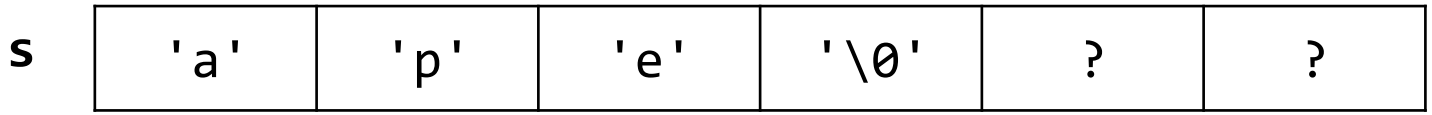
- Basta iniziare dalla prima cella (il cui indirizzo è noto)...
- ...E procedere in avanti finché non si incontra il terminatore

...Richiede sempre l'uso di un carattere addizionale

- Per memorizzare una sequenza di N caratteri...
- ...Servono N+1 celle!

Stringhe in C

Un array di N+1 caratteri può contenere stringhe più corte



Le celle dopo il terminatore:

- Concettualmente, sono vuote
- In pratica, contengono valori non controllabili

Definizione di Stringa

Una stringa si definisce esattamente come un array

Qualche esempio:

```
int s1[5];   // stringa di al più 4 caratteri
int s2[31];  // stringa di al più 30 caratteri
int s3[256]; // stringa di al più 255 caratteri
```

- Attenzione a ricordarsi di riservare una cella in più
- ...Servirà per il terminatore!

Inizializzazione di Stringhe

Si può inizializzare una stringa come si inizializza un array

Per esempio:

```
char s[4] = {'a', 'p', 'e', '\0'};
```

Oppure si può usare la forma compatta (con i doppi apici):

```
char s[4] = "ape";
```

- In questo caso il terminatore è aggiunto automaticamente

Inizializzazione di Stringhe

Se si usa la forma compatta si può omettere la dimensione

```
char s[] = "ape";
```

- Viene allocato il numero di celle necessario (i.e. 4)

Se si specifica una dimensione, deve essere abbastanza grande

```
char s[3] = "ape";
```

- Causa problemi! Il terminatore viene scritto fuori dai limiti

Esempio: Ricavare la Lunghezza (#caratteri)

```c
#include <stdio.h>
int my_strlen(char *s) {
    int len = 0;
    for(int i = 0; s[i] != '\0'; ++i) len++;
    return len;
}
int main() {
    char s[] = "ciao mondo";
    printf("Lunghezza: %d\n", my_strlen(s));
}
```

- Basta "scorrere" la stringa finché non si trova il terminatore
- Notate la condizione di avanzamento!

Esempio: Ricavare la Lunghezza (#caratteri)

```c
#include <stdio.h>
int my_strlen(char *s) {
    int len; // len accessible anche dopo il for
    for(len = 0; s[len] != '\0'; ++len);
    return len;
}
int main() {
    char s[] = "ciao mondo";
    printf("Lunghezza: %d\n", my_strlen(s));
}
```

- Si può riscrivere il codice usando una sola variabile
- Il corpo del ciclo for in questo caso è vuoto

Lettura e Scrittura di Stringhe

`scanf` e `printf` possono gestire le stringhe in modo semplificato

Per stampare una stringa si usa %s:

```
char s[] = "Gino";
printf("Ciao %s", s);
```

- La funzione stampa caratteri fino ad arrivare al terminatore

Per leggere una stringa si usa:

```
char s[81]; // preparo un array per la stringa
scanf("%s", s);
```

- Non ci vuole "&": "s" contiene già un indirizzo!

Lettura e Scrittura di Stringhe

Quando si legge una stringa con scanf e %s

- La funzione consuma eventuali spaziatori (i.e. ' ', '\t' o '\n')
- ...Quindi consuma e inserisce caratteri nella stringa
- ...Finché non incontra un altro spaziatore

```
char s[81];
scanf("%s", s);
```

- Se digitiamo " Cerutti Gino" (notate gli spazi iniziali)
- ...Nella stringa "s" viene inserito "Cerutti"

Lettura e Scrittura di Stringhe

Si può richiedere la lettura di caratteri fino alla fine della linea

- Si usa la specifica di formato "%[^\n]"
- La funzione consuma sempre eventuali spaziatori iniziali

```
char s[81];
scanf("%[^\n]", s);
```

- Se digitiamo " Cerutti Gino" (notate gli spazi iniziali)
- ...Nella stringa "s" viene inserito "Cerutti Gino"

Lettura e Scrittura di Stringhe

In alternativa si possono usare `gets` e `puts`:

```
char * gets(char *s);
```

Legge tutti i caratteri nel buffer di input fino a fine linea

- Il carattere di fine linea (i.e. '\n') non viene letto
- Non salta gli spaziatori iniziali (vengono inseriti nella stringa)
- Alla fine viene inserito un carattere di terminazione '\0'
- Restituisce un riferimento ad s in caso di successo
  - Non particolarmente utile
- Restituisce NULL in caso di fallimento
  - È il motivo per cui il tipo di ritorno è char*!

Lettura e Scrittura di Stringhe

In alternativa si possono usare `gets` e `puts`:

```
int puts(char *s);
```

Scrive i caratteri nella stringa

- Aggiunge un '\n' alla fine
- In caso di successo, restituisce un valore non negativo
- In caso di fallimento, restituisce EOF
  - EOF è una costante simbolica (definita e.g. in stdio.h)
  - Il nome sta per End Of File

Lettura e Scrittura di Stringhe

Quando si legge una stringa

...L'array di destinazione deve essere abbastanza grande!

```
char s[5];
scanf("%s", s);
```

Né scanf né puts effettuano alcun controllo sulle dimensioni

- Se digitiamo "supercalifragilistichespiralidoso"...
- ...La maggior parte dei caratteri viene scritto fuori dai limiti
- Come sapete, si tratta di un problema grave

Esempio: Copia di Stringhe

Si definisca la funzione:

```
char * my_strcpy(char *dest, char *src);
```

...Che copi il contenuto della stringa "src" nella stringa "dest"

- Si assuma (i.e. si dia per scontato) che src sia ben formata
- Si assuma che dest sia sufficientemente grande
- La funzione deve restituire l'indirizzo della prima cella di dest

Esempio: Copia di Stringhe

```c
#include <stdio.h>
char * my_strcpy(char *dest, char *src) {
    int i; // i serve anche dopo il ciclo for!
    for(i = 0; src[i] != '\0'; ++i)
        dest[i] = src[i];
    dest[i] = src[i]; // copio il terminatore!
    return dest;
}
int main() {
    char s[81], d[81];
    printf("Una parola: ");
    scanf("%s", s);
    my_strcpy(d, s);
    printf("Hai scritto: %s\n", d);
}
```

Esempio: Confronto di Stringhe

Si definisca la funzione:

```
int my_strcmp(char *s1, char *s2);
```

Che confronti alfabeticamente le due stringhe s1 ed s2

- Se s1 precede s2, deve restituire un numero negativo
- Se s1 ed s2 sono uguali, deve restiuire 0
- Se s1 segue s2, deve restituire un numero positivo

Convenzione:

- La stringa più corta ha la precedenza
- E.g. "arco" precede "arcobaleno"

Esempio: Confronto di Stringhe

Si definisca la funzione:

```
int my_strcmp(char *s1, char *s2);
```

Procedimento: scandire le due stringhe fintanto che:

- Nessuna stringa è terminata
- I caratteri nella stessa posizione sono uguali

Al termine, sfruttiamo il fatto che i caratteri in C siano numeri!

- Se s1[i] è più piccolo di s2[i], allora s1 precede s2
- '\0' ha un valore numerico più piccolo di ogni altro carattere

Esempio: Confronto di Stringhe

```c
#include <stdio.h>
int my_strcmp(char *s1, char *s2) {
    int i = 0;
    while(s1[i]!='\0' && s2[i]!='\0' && s1[i]==s2[i])
       i++;
    int res = s1[i] - s2[i];
    return res;
}

int main() {
    char s1[81], s2[81];
    printf("Due parole: ");
    scanf("%s%s", s1, s2);
    printf("Confronto: %d\n", my_strcmp(s1, s2));
}
```

Libreria string.h

Per lavorare con stringhe, il C fornisce la libreria string.h

Contiene diverse funzioni utili:

```
int strlen(char *s1);
```

- Lunghezza di una stringa (terminatore escluso)

```
char* strcpy(char *s1, char *s2);
```

- Copia la stringa s2 in s1

```
char* strcat(char *s1, char *s2);
```

- Concatena s2 ad s1

Libreria string.h

Per lavorare con stringhe, il C fornisce la libreria string.h

Contiene diverse funzioni utili:

```
int strcmp(char *s1, char *s2);
```

- Confronta s1 ed s2 (come nell'esempio)

```
char* strchr(char *s, char c);
```

- Restituisce un puntatore alla prima occorrenza del carattere c nella stringa s

```
char* strstr(char *s1, char *s2);
```

- Restituisce un puntatore alla prima occorrenza della stringa s2 nella stringa s1

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fondamenti di Informatica T

Andrea Acquaviva < andrea.acquaviva@unibo.it>
Michele Lombardi <michele.lombardi2@unibo.it
Andrea Borghesi <andrea.borghesi3@unibo.it>
Giuseppe Tagliavini <giuseppe.tagliavini@unibo.it>
Allegra De Filippo < allegra.defilippo@unibo.it>