TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

Macroarea di Ingegneria
Dipartimento di Ingegneria Civile e Ingegneria Informatica

Linguaggio C: Stringhe

Valeria Cardellini

Corso di Calcolatori Elettronici
A.A. 2019/20

Argomenti

- Dichiarazione di stringhe
- Inizializzazione e manipolazione di stringhe
- Funzioni di libreria per stringhe

Stringhe in C

- Una stringa è un vettore di caratteri
- Contiene la sequenza di caratteri che forma la stringa, seguita dal carattere speciale di fine stringa: `'\0'`
- Esempio:

```
char str[10] = {'p', 'r', 'o', 'v', 'a', '\0'};
```

- Esempio: il seguente vettore di caratteri non è una stringa perché non termina con `'\0'`

```
char non_str[2] = {'p', 'r'};
```


Inizializzazione di stringhe

- Una stringa può essere inizializzata anche utilizzando una stringa letterale

```
char str2[] = "prova"; oppure
char str2[6] = "prova";
```

  - In questi esempi l'array di caratteri è allocato staticamente
- E' possibile memorizzare una stringa in un array di caratteri allocato dinamicamente
  - Esempio:

```
char *buffer = malloc(20*sizeof(char));
```

  - In questo caso (come per tutti gli array allocati dinamicamente) non si può inizializzare l'array contestualmente alla sua creazione

Inizializzazione di stringhe

- Ad una stringa si può anche assegnare una stringa letterale

```
char *buffer2;
buffer2 = "prova";
```

  - Si assegna a `buffer2`, di tipo `char *`, la stringa costante `"prova"`, di tipo `char *` costante
- L'istruzione `buffer2[0] = 't';` non genera errore di compilazione, tuttavia causa un errore in esecuzione poiché si sta cercando di cambiare un carattere dichiarato costante

Codice sorgente: string_init.c

Inizializzazione di stringhe

- Inizializzazione di un vettore di stringhe (è un array di puntatori a char)
- Esempio

```
char *colori[4] = {"rosso", "giallo", "verde", "blu"};
```

  - E' un vettore di quattro puntatori a quattro stringhe costanti
  - E' equivalente ad inizializzare i quattro puntatori separatamente:

```
char *colori[4];
colori[0] = "rosso";
colori[1] = "giallo";
colori[2] = "verde";
colori[3] = "blu";
```

Input/output di stringhe

- Per stampare una stringa si deve utilizzare la specifica di formato %s
  - Esempio

```
printf("%s\n", stringa1);
```

  - Vengono stampati tutti i caratteri fino al primo '\0' escluso

Input/output di stringhe

- Per leggere una stringa si utilizza la specifica di formato %s
- Esempio

```
char buffer[40];
scanf("%s", buffer);
```

  - Vengono letti da input i caratteri in sequenza fino al primo carattere di spaziatura (spazio, tabulazione, interlinea, ecc.)
  - I caratteri letti vengono memorizzati nell'array buffer, terminati dal carattere '\0'
- Osservazioni:
  - L'array deve essere sufficientemente grande da contenere tutti i caratteri letti
  - Non si usa `&buffer` ma direttamente `buffer` (perché `buffer` è già un indirizzo)

Manipolazione di stringhe

- Per manipolare una stringa si deve accedere ai singoli caratteri
- Esempio

```
for (i = 0; buffer[i] != '\0'; i++)
  printf("%c\n", buffer[i]);
```

- Esempio: Confronto di uguaglianza tra due stringhe
  - Non si può usare `str1==str2` perché così si confrontano i puntatori e non le stringhe
  - E' necessario scandire le due stringhe e confrontare i singoli caratteri
  - Nella standard library funzione `strcmp()`

Esempio: manipolazione di stringhe

- Confronto tra 2 stringhe

```
int stringcmp1(char  * str1, char *str2) {
  int i = 0;
  while (str1[i] == str2[i] && str1[i] != '\0' && str2[i] != '\0') {
    i++;
  }
  return str1[i] == '\0' && str2[i] == '\0'; // 1 se uguali, 0 altr.
}

int stringcmp2(char  * str1, char * str2) {
  while (*str1 == *str2 && *str1 != '\0' && *str2 != '\0') {
    str1++;
    str2++;
  }
  return *str1 == '\0' && *str2 == '\0';
}
```

Codice sorgente: string_comp.c

Stringhe e caratteri

- Caratteri e stringhe sono diversi e non vanno confusi:
  - un carattere è in realtà un intero
  - una stringa è un vettore di caratteri che termina con il carattere '\0'
  - una variabile di tipo stringa è in realtà un puntatore al primo carattere del vettore
- Esempio

```
char c = 'a'; // carattere
char *s = "a"; // puntatore a stringa costante "a"
char v[] = "a"; /* vettore di 2 char inizializzato a
                   {'a', '\0'} */
printf("%lu %lu %lu\n", sizeof(c), sizeof(s), sizeof(v));
```

  - Stampa: 1 8 2

Funzioni di libreria per l'input/output di stringhe e caratteri

- E' necessario `#include <stdio.h>`
- Input e output di caratteri

`int getchar(void);`

  - Legge il prossimo carattere da standard input e lo restituisce

`int putchar(int c);`

  - Invia c allo standard output
  - Restituisce EOF se si verifica un errore

Funzioni di libreria per l'input/output di stringhe e caratteri

- Input e output di stringhe

```
char *gets(char *str);
```

  - Legge i caratteri da standard input e li inserisce in `str`
  - La lettura termina con `'\n'` o `EOF`, che non viene inserito in `str`
  - La stringa viene terminata con `'\0'`
  - Attenzione: non c'è alcun modo di limitare la lunghezza della sequenza immessa, lo spazio allocato potrebbe non bastare (buffer overflow); l'uso di `gets` è sconsigliato

```
int puts(const char *str);
```

  - Manda la stringa `str` in standard output inserendo `'\n'` alla fine

Funzioni di libreria per l'input/output di stringhe e caratteri

- Input e output di stringhe da/su stringhe

```
int sprintf(char *str, const char *format, ...);
```

  - Come printf ma scrive in str

```
int sscanf(char *str, const char *format, ...);
```

  - Come scanf ma legge da str

Funzioni di libreria per la manipolazione di stringhe

- E' necessario `#include <string.h>`
- Funzioni di copia

`char *strcpy(char *dest, const char *src);`

  - Copia la stringa `src` nel vettore `dest`
  - Restituisce il valore di `dest`
  - `dest` dovrebbe essere sufficientemente grande da contenere `src`
  - Se `src` e `dest` si sovrappongono, il comportamento di `strcpy` è indefinito

- Esempio

```
char a[10], b[10];
char *x = "Hello", *y = "world";
strcpy(a, x); strcpy(b, y);
puts(a); puts(b);
```


Funzioni di libreria per la manipolazione di stringhe

`char *strncpy(char *dest, const char *src, size_t n);`

  - Copia massimo `n` caratteri della stringa `src` nel vettore `dest`
  - `size_t` è il tipo del valore restituito da `sizeof`
  - Restituisce il valore di `dest`
  - Attenzione: il carattere `'\0'` finale di `src` viene copiato solo se `n` è ≥ della lunghezza di `src+1`

Funzioni di libreria per la manipolazione di stringhe

- Funzioni di concatenazione

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

  - Concatena la stringa `src` a quella nel vettore `dest` (il primo carattere di `src` si sostituisce a `'\0'` di `dest`) e termina `dest` con `'\0'`
  - Restituisce il valore di `dest`
  - `dest` dovrebbe essere sufficientemente grande da contenere tutti i caratteri di `dest`, `src` e `'\0'` finale
  - Se `src` e `dest` si sovrappongono, il comportamento di `strcat` è indefinito

```
char *strncat(char *dest, const char *src, size_t n);
```

  - Per concatenare al più `n` caratteri di `src` a `dest`

Funzioni di libreria per il confronto di stringhe

- Funzioni di confronto

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

  - Confronta le stringhe `s1` ed `s2`
  - Restituisce:
    - ✓ 0 se s1 = s2
    - ✓ un valore < 0 se s1< s2
    - ✓ un valore > 0 se s1 >s2
  - Confronto di tipo lessicografico: caratteri confrontati uno ad uno ed il primo carattere diverso (o la fine di una delle due stringhe) determina il risultato

```
int strncmp(const char *s1, const char *s2, size_t n);
```

  - Confronta al più `n` caratteri di `s1` ed `s2`

Funzioni di libreria per la ricerca di stringhe

- Funzioni di ricerca

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

  - Individua la prima occorrenza di `s2` in `s1`
  - Se trova la stringa, restituisce un puntatore alla posizione di `s2` nella stringa `s1`; altrimenti `NULL`

Funzioni di libreria per la conversione di stringhe

- E' necessario `#include <stdlib.h>`
- Convertono le strighe formate da cifre in valori interi ed in virgola mobile.
- Funzioni `atoi`, `atol`, `atof`

```
int atoi(const char *str);
```

  - Esempio:

```
int n = atoi("123");
```

Esempio: sottostringa

- Determinare se una stringa è contenuta in un'altra e quante volte

```
int substring(char *substr, char *str, int *n) {
  int i = 0, j, continua;
  *n = 0;
  while (str[i] != '\0') {
    continua = 1;
    j = 0;
    while (substr[j] != '\0' && str[j+i] != '\0' && continua) {
      if (substr[j] != str[j+i]) continua = 0;
      j++;
    }
    if (continua && substr[j] == '\0') (*n)++;
    i++;
  }
  return *n > 0;
}
```

Codice sorgente: substring.c

Esempio: sottostringa

- Determinare se una stringa è contenuta in un'altra e quante volte usando la funzione `strstr()`

```
int substring2(char *substr, char *str, int *n) {
  int len;
  char *s;
  *n = 0;
  len = strlen(substr); /* lunghezza della sottostringa */
  if (len > 0) {
    s = str;
    while ( (s = strstr(s, substr)) != NULL ) {
      (*n)++;
      s += len;
    }
  }
  return *n > 0;
}
```

Codice sorgente: substring.c